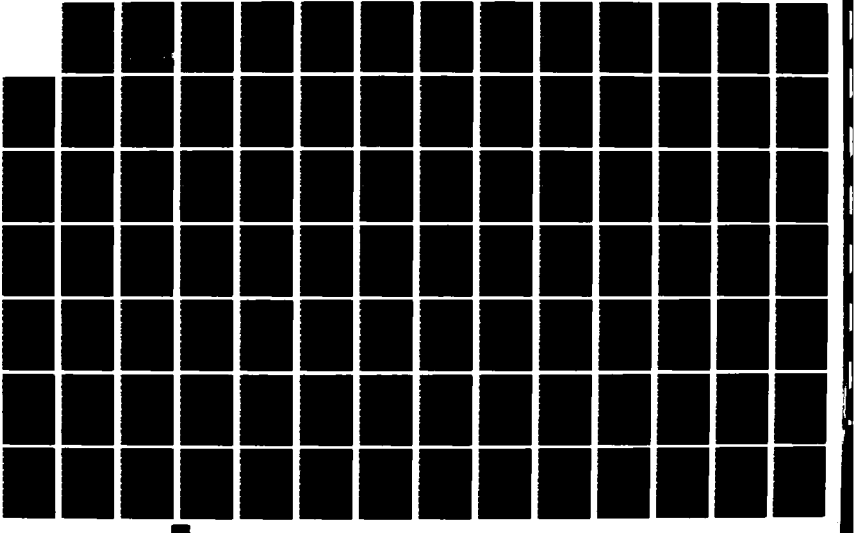
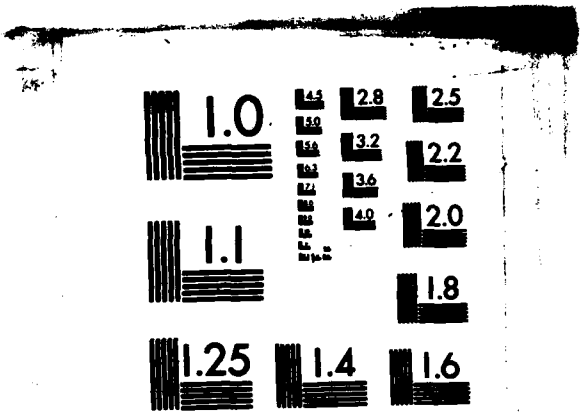
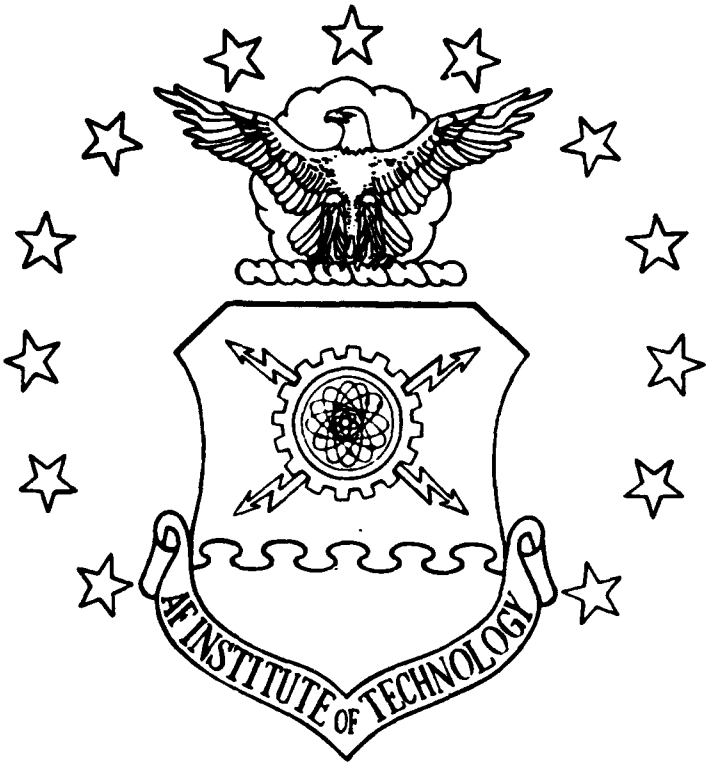
AD-A138 268 APPLICATIONS OF MULTIVARIATE STATISTICAL TECHNIQUES FOR COMPUTER PERFORMANCE EVALUATION(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 1/3

UNCLASSIFIED G L BRUNDIDGE DEC 83 AFIT/GCS/EE/83D-4 F/G 12/1 NL







APPLICATIONS OF MULTIVARIATE STATISTICAL TECHNIQUES FOR COMPUTER PERFORMANCE EVALUATION

THESIS

AFIT/GCS/EE/83D-4 Gregory L. Brundidge

Capt USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/EE/83D-4

APPLICATIONS OF MULTIVARIATE STATISTICAL TECHNIQUES FOR COMPUTER PERFORMANCE EVALUATION

THESIS

AFIT/GCS/EE/83D-4 Gregory L. Brundidge

Capt USAF

Approved for public release; distribution unlimited

AFIT/GCS/EE/83D-4

APPLICATIONS OF MULTIVARIATE STATISTICAL TECHNIQUES FOR COMPUTER PERFORMANCE EVALUATION

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Gregory L. Brundidge, B.S.

Capt USAF

Graduate Computer Science

December 1983

Approved for public release; distribution unlimited.

Preface

This study was motivated by the continuing need for practical and economical means to evaluate computer systems. The multivariate statistical techniques examined are still new to the CPE environment. However, as more analysts become familiar with the possibilities provided by the data analysis tools, usage should increase.

It is assumed that the reader has a working knowledge of basic univariate statistics and is familiar with basic principles underlying multivariate analysis. Emphasis is placed on applications for the techniques and theoretical development has been kept to a minimum. The interested reader should keep in mind that approaches and applications used in this study were largely influenced by the system being evaluated and the availability of all the necessary resources to perform the various analyses. Therefore, prior to attempting similar analyses one should make a good assessment of the resources required. Special thanks are in order for my advisor, Dr Hartrum and my readers Maj Joseph W. Coleman, Capt Brian W. Woodruff, and Dr Gary B. Lamont. Their inputs and assistance made this study a valuable learning experience. I would also like to give a special thank you to my wife who provided me with the constant everyday support necessary to complete this study.

Contents

List of Figures

List of Tables

Abstract

In many situations the computer performance evaluation (CPE) analyst has collected an abundance of computer system performance data from the target system's accounting files and software monitors. Traditionally, regression analysis provided the primary means of examining CPE data sets, with the emphasis being on modeling specific workload and performance parameters. Multivariate analysis techniques provide the analyst with additional analysis tools for the examination of relationships, dimension, and structure of large amounts of data. This study examines possible CPE applications for four multivariate analysis techniques. The techniques studied include: Canonical Correlation, Factor Analysis, Discriminant Analysis, and Cluster Analysis. Also included in the study was the use of ordinary least squares regression modeling and ridge regression modeling, to exemplify the traditional problems encountered with use of regression analysis. Depending on the performance evaluation requirements, one or more of the multivariate techniques or ridge regression could be used to perform a preliminary or supplementary CPE data analysis.

CHAPTER I

INTRODUCTION

Background

Because of the constant need for timely processing of information and data, an increasing number of the U. S. Air Force's mission essential and routine operations are being handled by computer. As applications for the computer expand, so do the requirements for efficient operating computer systems. Consequently, peformance evaluation of Air Force computer systems has become an area of increasing interest. While numerous tools are available for evaluating computer system performance, many are not practical for frequent or routine use. Due to cost, additional equipment requirements, complexity and time required, many evaluation tools are only practical after performance problems are known to exist. Thus, instead of simulation models, complex analytical analyses, or time consuming system capacity studies, an alternate or supplemental means of evaluating performance is the examination of system-generated data from accounting files and activity monitors. To perform an evaluation of this type, requirements include access to, and knowledge and understanding of an applicable data analysis technique(s).

Multiple linear regression analysis has been a useful means of analyzing computer performance based on

system-generated performance data. However, its use is limited due to the assumptions of non-multicollinearity and normality that must hold for the independent variables and error terms. Thus, a requirement exists for further examination of techniques that would enable computer performance evaluation (CPE) based on empirical data produced by the computer under study (target system).

To examine additional data analysis techniques, a recent AFIT/EN thesis (Ref 11) was completed in which six statistical techniques were evaluated for potential CPE applications. Using regression analysis as a basis for comparison, the techniques investigated included 1) ridge regression, 2) automatic interaction detection (AID), 3) cluster analysis, 4) canonical correlation analysis, 5) factor analysis and 6) discriminant analysis. Reported results suggested that the techniques have varying levels of application to CPE. Since the tests performed to derive these results used simulated performance data, further examination of the techniques using real performance data from a target computer should provide an increased understanding of possible CPE applications. Applying the techniques to real performance data would allow obtained information to be evaluated in light of the observed actual peformance of the target system. Thus, as a follow-on study five of the statistical techniques listed above were applied to real system-generated performance data in an

attempt to evaluate the target computer's performance and the usefulness of each technique as a CPE tool.

Problem

The problem investigated in this thesis effort was the partial performance evaluation of AFIT's VAX 11/780 scientific support processor (SSC) using information gained from applying the six data analysis and multivariate techniques listed above. The performance evaluation was partial because system performance was evaluated only to the degree allowed by the multivariate techniques being studied. It was hoped that the information gained about system performance by applying the techniques would enable a specific definition of each of the techniques' possible application(s) in a CPE environment. Since no routine CPE practices (i.e. routine analysis of software monitor or accounting file summary output) for the SSC existed at the time this study was conducted, secondary objectives of the evaluation included:

1) Providing background information for identification of possible performance improvement areas.
2) Providing measurements and descriptions of the workload currently processed by the system.
3) Providing a means of compiling performance and use trends for use in forecasting future system

demands.

In addition to being used as a means to establish CPE applications for the multivariate techniques, the above objectives were chosen to also provide the basic framework for establishing routine CPE practices to monitor performance of AFIT's VAX 11/780. It had been noted that certain workloads resulted in a noticeable degradation in VAX performance. However, it had not been determined exactly what aspects of the workload caused the degraded performance. As this performance evaluation progressed, it was hoped that application of the multivariate techniques would aid in identifying the causes of degraded performance. Also, where possible, recommendations for future prevention of the discovered causes of degradation were made.

Scope

One of the primary goals of this thesis was to evaluate the usefulness of selected multivariate techniques as CPE tools. Therefore, only these techniques were used to do the performance evaluation. Consequently, the performance evaluation was not a total evaluation in that not all available CPE tools, such as simulation and analytical techniques, were used. Though other multivariate techniques exist, only those mentioned in the problem statement were evaluated. The study was conducted using

data from a VAX 11/780 running the UNIX operating system. Consequently, in some instances, results reflected performance characteristics which were unique to this system configuration and may differ from results of similar tests on different systems or similar systems using a different operating system. However, recommended CPE applications for the techniques being studied should be similar in many performance evaluation environments.

Assumptions

Throughout this thesis effort, no attempt was made to improve or modify the software implementations of the statistical techniques under study. It was assumed that current literature on the application of each of the techniques was valid and that no further development of the application methods was necessary. Also, all techniques used were assumed to be correct as they were implemented on the VAX or CYBER.

Summary of Current Knowledge

With the exception of multiple linear regression, use of the multivariate techniques considered in this thesis, for CPE purposes, has been limited. The primary efforts thus far have been two theses by Magavero (Ref 11) and Stover (Ref 18). In Magavero's effort, each of the six techniques was applied to selected data sets which were generated using Computer Performance Evaluation Simulator

(CPESIM), a computer system simulation program. His primary goal was to determine how the techniques could be used in CPE. Realizing that each technique had specific applications and theoretical limitations, Magavero applied them to data sets which would allow him to determine how well a technique described performance factors present in the data. In the earlier thesis by Stover (Ref 18), three techniques were examined, multiple linear regression, automatic interaction detection and ridge regression. While her approach was similar to Magavero's, she concentrated her efforts in building empirical models and then rating the techniques on how well they modelled the actual system performance. Like Magavero, Stover also used data generated by running CPESIM.

Other literature includes a paper by Hartrum and Thompson entitled "The Application of Clustering Techniques to Computer Performance Modeling" (Ref 8) and a paper by A. Agrawala and J. Mohr entitled "Some Results of the Clustering Approach to Workload Modeling." (Ref 1) Both of these papers were reports on results derived from examining selected performance factors using cluster analysis.

In both of the referenced thesis efforts, simulated performance data was used. The papers mentioned addressed only one of the techniques. While other literature was available, usually only one technique was considered and reports were only on test results, with little or no

information addressing possible CPE applications. Thus, for the reasons previously stated, extension of Magavero's effort to include the use of real data was expected to provide a better understanding of the applications that are possible.

Standards

Where applicable, all interpretations of statistical results were evaluated via use of the appropriate statistical inference tests. For example, confidence intervals were set at a minimum of 90% and an evaluation of the derived results was done based on the size of the calculated confidence intervals. The assumptions necessary to apply each of the techniques was also used to evaluate the usefulness of the interpreted results.

Approach

With respect to the performance evaluation aspect of this thesis effort, initial investigation was directed at system understanding and familiarization. A study of the VAX architecture and the UNIX operating system was conducted. The study was expected to provide the necessary understanding of the hardware configuration used at AFIT and the function of the UNIX operating system running on the AFIT VAX. The study revealed the performance measurement tools that were available in the hardware and software of the target system.

A survey of operating procedures was also done to

provide a good understanding of operator, analyst, and system interaction. Since no CPE program currently existed for the AFIT VAX, the survey included any CPE practices that were currently being used for any of the other AFIT systems.

After system familiarization was complete, performance data available on the system was studied and analyzed for identification of performance factors and related performance parameters. Information gained from this study was used to identify the performance and workload measures that would be used when applying the multivariate techniques.

The above preliminary studies along with the actual generation of data sets and analysis of multivariate techniques were done in ten phases, as follows:

1) A study of the performance characteristics of the VAX and the current AFIT operational practices was conducted to gain insight into the performance environment from which the empirical data was collected.
2) The data set collection procedures were selected to ensure that the data sets used for statistical tests reflected as closely as possible the desired computer performance environment.
3) Sets of performance data from the computer chosen for evaluation were analyzed to establish data

relationships, identify the criterion and predictor variables, and to determine any other parameters necessary for properly applying the statistical techniques.

4) Each statistical technique was applied to the selected data sets and results were recorded.

5) The results were interpreted to determine what performance evaluation information was present and then an attempt was made to evaluate the computer system based on the obtained information.

6) When possible, statistical inference tests were obtained. Evaluation was based on the level of confidence that could be placed in the interpretations reported and the assumptions required for application of the technique.

10) Finally, based on the evaluation tests that were done, recommendations were made as to the potential applications of each of the statistical techniques used. All significant performance evaluation information obtained for the target system was also reported.

Support

In order to accomplish the preliminary studies, assistance from the system programmer and system operators was necessary to gather the performance data required for applying the statistical techniques. Since some of the

techniques being studied were programs or part of statistical packages on the Aeronautical Systems Division (ASD) CYBER computer, assistance was also required to transfer data between machines. Because there was no direct communication line between the two computers, the anticipated transfer technique was to involve writing generated data sets to tape and then transferring the tapes to the CYBER. Initially, attempts were made to write the collected VAX data to tape and then read the data from tape onto disk files on the CYBER. However, because of difficulties encountered in getting the CYBER to read the VAX generated tapes, data transfer was done by down loading the data, via communication lines, to a microcomputer diskette and in turn, transferring it to a CYBER disk file.

CHAPTER II

MULTIVARIATE ANALYSIS TECHNIQUES

As stated in the introduction, the techniques studied included multiple linear regression, canonical correlation analysis, ridge regression, cluster analysis, factor analysis and discriminant analysis. Based on results of past applications, each technique is to some degree capable of describing specific characteristics about interrelationships that exist in a multivariate data set. This chapter provides a brief discussion of the theoretical background and anticipated applications for each of the techniques being studied.

Multiple Linear Regression

Multiple linear regression provides the analyst with a means to investigate the strength and nature of relationships among interval scaled variables. It allows construction of a functional relationship between the variables that can be used for explanative and predictive purposes. Variables fall into two classes; criterion or dependent variables, the variable for which a value is to be predicted; and predictor or independent variables, those variables used to do the prediction. The multiple linear

regression model has the following form:

$$y_i = b_0 + b_1 x_1 + \ldots + b_j x_j + e_i \qquad (1)$$

In the model, y_i represents the value of the criterion variable in the i-th of n cases. The b_j values represent coefficients which indicate the proportional relationships between the j-th independent variable and the criterion variable. The error term e_i represents the difference between the exact value of y_i and the predicted value of y_i. Three statistical assumptions about the error terms allow the analyst to construct hypotheses based on the derived model. These assumptions are: 1) the distributions of the error terms are identical, 2) the error terms themselves are independent, 3) the error terms are distributed as a normal with mean zero and variance (sigma-square).

Should a multiple linear regression model exist for a given population, the values of the coefficients can be estimated. The most widely used method for estimating the coefficients of the model is the least squares technique. The objective in using least squares is to find coefficient estimates which minimize the sum of squared differences between the observed values of the criterion variable, y_i, and predicted values of the criterion variable y_i. In matrix terms the estimated model becomes:

$$\underline{y}' = \mathbf{X} * \underline{B}' + \underline{e}' \qquad (2)$$

Where $\underline{y}'$ is an n element column vector, $\underline{B}'$ is a k + 1 element column vector and $\mathbf{X}$ is an n x k+1 matrix of variable values.

Proper use of the multiple linear regression model requires that four major assumptions be met.

1. The variance of e_i must be constant for all cases in the model. Violation of this assumption is called hetereoscedasticity, and results in unreliable tests of the null hypothesis that all coefficients are equal to zero.
2. The error terms must be independent. Violation of this assumption is called autocorrelation, and again results in unreliable inference tests on the coefficients.
3. The error terms must be normally distributed about the n-dimensional hyperspace represented by the model.
4. All independent variables must be independent of each other. Dependence among independent variables is called multicollinearity and results in unstable regression coefficients.

All of the conditions which result from violation of

the above assumptions result in unreliable inference tests on the coefficients which have been determined using the least squares technique.

Ridge Regression

In regression models built from real world data it is likely that some of the independent variables will be related. Thus, multicollinearity will exist to some degree. Ridge regression is a variation of the ordinary least squares regression which attempts to negate the effects of multicollinearity by decreasing the variance associated with each coefficient. This regression technique uses biasing in the calculation of the regression coefficients to compensate for the interrelationships that exist between the independent variables.

The ridge regression algorithm solves for a vector of coefficients using the following equation:

$$b^{*} = (X'X + kI)^{-1} X'\underline{y} \qquad (3)$$

X represents the standardized data matrix and k is the bias value which is incremented in discrete steps over a succession of iterations to produce a better set of estimated coefficients.

The output of the ridge regression algorithm used in this study included normalized and unnormalized estimated

coefficients, a graphical "ridge trace," and variance inflation factors (VIFs), for each value of k. The ridge trace depicts graphically the stabilization of the calculated coefficients as the value of the bias term, k, increases. VIFs are the diagonal elements of the inverse X'X matrix and provide a measure of multicollinearity. VIFs are calculated using (Ref 9:124):

$$VIF_i = 1/(1 - r_2) \text{ (with k=0,see Eq 3)} \quad (4)$$

R is the multiple correlation coefficient between the given dependent variable and all other independent variables.

Ridge regression output from the program used in this study provided values of k that render improved coefficients. The selection of the best value of k may be based on certain heuristics. Common non-graphical heuristics are 1) all VIFs less than 10, and 2) all improper signs changed. Regardless of the heuristic used, the objective will be to choose as small a value of k as possible, thereby minimizing the amount of bias introduced into the calculations. The most popular graphical technique for selecting k values is examination of the ridge trace. In this case the value of k is chosen such that the values are fairly stable (not changing rapidly).

Canonical Correlation

In analyzing multivariate data it may be required to examine relationships between selected sets of variables. Canonical correlation is a multivariate analysis technique which enables the analyst to examine variables which are themselves described by two or more variables. The technique involves the derivation of a canonical variate for each set of variables such that the correlation between the sets is maximized.

The method is designed for use not only when the independent variables take a linear additive form, but also where there may be a set of two or more dependent variables in linear additive form. The method, however, assumes that the two sets of variables are distinct. Canonical correlation is therefore applied in the situation where in addition to the set of variables which serve as predictors, a set of variables exist which serve as criteria. The objective, as stated earlier is to build a linear model between these two sets of variables, where the coefficents in both sets are to be determined on the basis of obtaining the maximum correlation between the two sets of variables.

In the analysis, one would effectively have compound independent and dependent variables X^* and Y^* described by:

$$X^* = b_1 x_1 + b_2 x_2 + \ldots + b_k x_k \qquad (5)$$

$$Y^* = a_1 y_1 + a_2 y_2 + \ldots + a_p y_p \quad (6)$$

In performing the canonical correlation analysis it is the a_i s and b_i s that must be found simultaneously so that the correlation between X^* and Y^* (referred to as super variables) is maximized (Ref 14:Ch 5). The resulting correlation is termed the canonical correlation.

Factor Analysis

When the analyst is faced with a data set containing many variables, analysis of variable relationships using regression techniques may be hampered by strong inter-variable relationships that exist in the data set. As stated in the section on regression, existence of multicollinearity causes poor regression coefficient estimation and thus reduces the explanatory power of the regression model. Factor analysis provides a means for the analyst to effectively reduce the dimensionality of the data set while simultaneously deriving a set of truly independent factors which represent the original set of variables. The computational procedure involved in performing factor analysis is:

1. Find all eigenvalues of the k x k matrix X'X (where X is the n x p data matrix) and arrange in order of magnitude so that lambda$_j$ represents the largest eigenvalue.

2. Find the normalized eigenvector associated with each eigenvalue. These eigenvectors can be arranged in a k x k matrix denoted **A**. Each column in A is denoted $\underline{a}_j$ so that $\underline{a}_j$ is associated with lambda$_j$. The **A** matrix is the factor pattern.
3. The fraction of variance explained by the j^{th} component is determined from lambda$_j$ and the cumulative fraction of variance calculated by summing these values (Ref 14:Ch 6).

In a broad sense, factor analysis is a method for reformulating a set of observed independent variables into a new set (usually fewer in number, but never more in number) of independent variables, such that the latter set has certain desired properties specified by the analyst (Ref 17:237-260). Two commonly used factor analysis techniques are principal component analysis and classical factor analysis. Principal component analysis is the search through data to find factors or components that may reduce the dimensions of variation and assign possible theoretical meaning. On the other hand, classical factor analysis starts with the hypothesis of a model and tests it against the data to see if it fits (Ref 14:Ch 6).

Discriminant Analysis

When studying multivariate data, the requirement may exist to determine whether or not differences exist between groups within a multivariate population. In this situation the analyst may want to classify observations in the multivariate population into two or more distinct groups based on measureable differences. Typical questions asked include:

1) Are the two groups significantly different with respect to their multivariate descriptions?

2) What role do the variables for which measurements have been obtained play in separating the groups?

3) If levels for the variables are known for a new observation, to which group does the case belong?

The questions show that a full discriminant analysis will actually consist of three sub-analyses of the multivariate population. One step will involve the attempt to discriminate between groups in a population. The second step involves determining whether or not an inter-group difference really exists within the population. Finally, the third step involves an attempt to classify new observations into the derived groups of the population.

Cluster Analysis

In multivariate data where relationships exist between the variables, it is possible and at times desirable to group variables based on these relationships for exploratory

purposes. Cluster analysis provides the analyst with capability to perform this type of grouping. The term cluster analysis actually refers to a variety of methods for grouping data by observation. All of the methods use some measure of similarity and can be described as either hierachical or non-hierarchical. Clustering algorithms exist which allow clustering of either variables, observations, or both. Common measures of similarity between objects include the Euclidean distance, or Mahalanobis distance as distance metrics and the product-moment correlation as a shape metric (Ref 2: Ch 1). Clustering performed with a hierarchical algorithm will start with all variables in one cluster and through an iterative process, break the variables into smaller and smaller clusters based on a specified similarity measure. The iteration continues until each variable is in a cluster of its own or a specified number of clusters has been reached. Because of the computations required to compute the similarity measures for each variable in each iteration, implemented hierarchical algorithms usually require extensive processing resources (memory and cpu)for large data sets. On the other hand, non-hierarchical clustering techniques are less demanding of computer

resources and initially consider each variable or observation as being in a cluster of its own. Through what is usually a stepwise process, variables or observations are then clustered based on a specific similarity measure (Ref 2:Ch 7,8,9).

Differences in derived clusters can be evaluated based on the already mentioned similarity measures or by use of analysis of variance (ANOVA). Studies of how well collected performance data reflects changes in system activity due to altered performance parameters provide a possible CPE application for cluster analysis (Ref 8). Whether or not a hierarchical or non-hierarchical clustering technique is used will in most cases be determined by the nature of the data set being analyzed. If the data consists of variables which measure separate but related system activities (i.e. for a given job, the cpu utilization, memory required, memory used, number of disk I/Os) then a hierarchical technique could be used to examine how these variables cluster for a given set of jobs. This in turn may provide insight into the relationship specific jobs have to measured performance and workload parameters. However, if variables in the data reflect joint activity (i.e. concurrent cpu, memory and disk activity obtained from a software or hardware monitor) a non-hierarchical technique could be used to observe how observations across a spectrum of system activity would cluster. If the data could then be

related to some aspect of the systems processing environment such as a specific type of processing (i.e. simulations or text formatting), the derived clusters of observations (or cases) may provide insight into the workload being generated by a specific type of processing.

While the above uses of clustering in CPE are possible, the nature of clustering provides the analyst with essentially a non-restrictive tool for examining relationships between variables and observations. Thus, of all the techniques being examined, cluster analysis would give the analyst the most flexibility in application and in interpreting the results.

CHAPTER III

EXPERIMENTAL DESIGN CONSIDERATIONS

The study of the application and use of the multivariate techniques for CPE purposes required a four phased experimental design. The first phase consisted of familiarization with system architecture and operation. Results of this phase are presented in detail in Appendix A. The second phase involved a review of possible sources of workload and performance data. The third phase consisted of collecting workload and performance data using selected software monitors and accounting data, and preparing the collected data into data sets suitable for multivariate analysis. The fourth phase consisted of application of the multivariate techniques and analysis of the results. With the exception of phases one and four, actions taken and results for each phase are discussed in this chapter. Also, as an extension to the discussion of system familiarization presented in Appendix A, a section on the current system workload is presented in this chapter. While overall analysis objectives for each of the multivariate techniques will be discussed in general, specific objectives of application and results are presented in subsequent chapters.

Current System Workload

The AFIT VAX 11/780 Scientific Support Computer (SSC) is faced with a very dynamic workload environment. Initially, the intended uses for the SSC included primarily scientific/engineering applications, such as general applications programming, simulation, mathematical and statistical analysis, and advanced data base applications. However, until a policy change which took effect during the course of this study, the SSC workload also included a significant amount of text formatting used for preparing papers and audiovisual aids. While graphics capabilities were also intended initially, heavy use of the SSC for graphics processing also represented a significant portion of the workload. Figure III-1 shows the relative sizes of components which make up a typical SSC workload.

Over the course of a year, quarter, week or day the degree to which any of the above types of processing contributed to the overall system workload was not constant. The academic environment which the SSC supports is primarily responsible for the highly varying workload. Based on observation, yearly workload tends to be greater during winter, spring, and summer quarters. Quarterly workload rose and fell with class project due dates which resulted in heavier workloads near weeks five and ten.

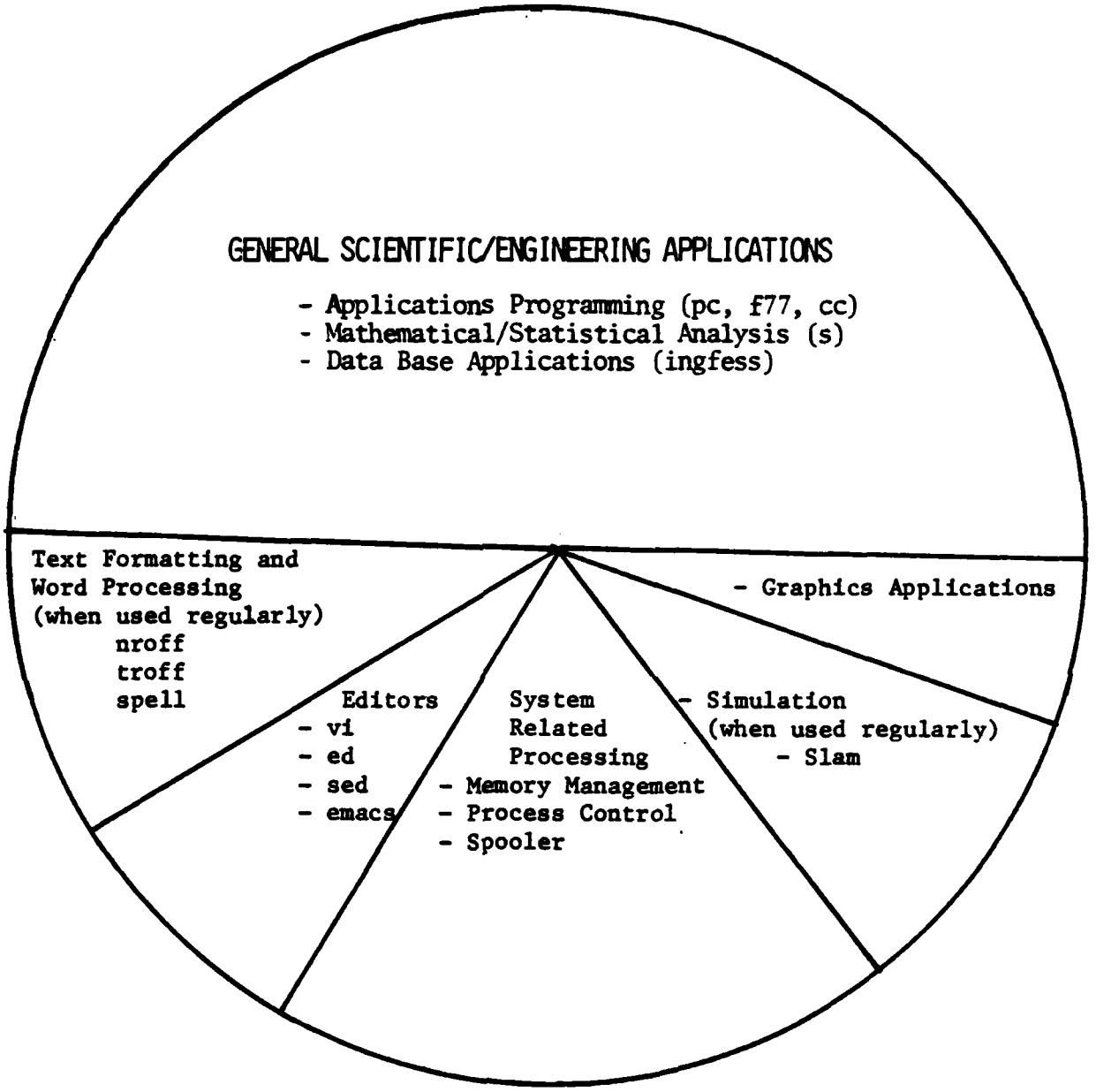


Figure III-1 Typical SSC Workload Components

Weekly workload was greatest during prime duty hours (0800-1700) and in some cases early evening hours (1800-2200). While these relative trends in periodic workload show an aspect of the high variability in SSC processing environment, it must be stressed that at any given time, components of the workload will usually not be the same.

At the time this study was being conducted it was apparent that certain workloads resulted in significant degradation of system performance. Thus, from a performance evaluation viewpoint, a system bottleneck did exist and a goal of the analyses to be conducted was to identify, if possible, the causes of the bottleneck and present possible solutions. However, bottleneck detection and performance improvement were not the primary objectives of this research and these issues were only handled to the degree that the multivariate analysis techniques allowed.

Phase Two - Data Sources

The SSC runs Berkeley UNIX Version 4.1 and comes equipped with a number of software monitors and accounting routines. A detailed discussion of the primary routines and monitors available is presented in Appendix B. For the purposes of this study, three software monitors were used in addition to selected data from the system accounting file to form a multivariate data set, which represented

overall system activity. The software monitors used included:

1. 'vmstat' - an interval driven monitor which samples virtual memory, disk and cpu activity.

2. 'ps' - a system command which can be used in a loop to provide periodic data on current process table entries. PS was used in conjunction with 'wc' (a counting utility) to provide a count of all (logged in users) processes.

3. 'df' - a system command which when used in a loop can provide periodic disk/file system utilization data.

The monitors used are summarized in Table III-1.

Table III-1

Software Monitor Description

MONITOR	FUNCTION
vmstat	-- interval driven monitor which reports overall system activity.
ps	-- process status monitor which reports disposition of process current being executed on the system.
df	-- disk utilization monitor which reports utilizations for all mounted file systems (real and logical).

Vmstat provided the most comprehensive view of system activity. Vmstat output includes twenty-two items of sampled system activity which together form a useful set of workload measures. Some of the items, such as the number of runnable processes, anticipated short-term memory short fall, and pages scanned by clock algorithm per second are also indirect measures of performance because increases in these items indicate that the system is not performing at a level necessary to handle its current workload.

The accounting data used was taken from the system activity accounting file /usr/acct. Included was summary data on executed processes execution time, cpu operation, I/O activity and memory usage. Also extracted were counts of selected commands that were processed to characterize the workload by the types of processes being run. Monitored commands were grouped into fifteen categories which represented specific types of processing. Finally, the average total execution time required for the 'date' command was extracted to be used as a benchmark for a response time indicator. Ideally, the time required to execute the 'date' command should be less than .3 seconds. Thus, increases in execution time for the command provided an indication of degraded performance. The data items which made up the composite data file were selected because together they provided a combination of workload measures which would allow different aspects of system performnce to

be analyzed.

Phase Three - Data Collection and Preparation

Once the data sources were selected, data collection and preparation routines were developed. For data collection, UNIX scripts (executable files of UNIX commands) were used and executed beginning at midnight using the UNIX 'at' utility. The software monitor routines were initiated and ran for a 24-hour period while initiation of the accounting data collection routines extracted the previous 24-hour period entries in the system accounting file.

Software monitors were run at 5-second interval sampling rates with collected data being appended to a data file every five minutes. Because of the high sampling rate, software monitor data files grew to over 2 megabytes within the 24-hour monitoring period. To avoid filling the /usr file system, data files were dumped to tape each day. Also dumped were the raw accounting data files which had been extracted from the system accounting file.

Once monitor and accounting data files had been collected, preparation of data sets primarily involved reduction of the voluminous monitor data files and summarization and extraction of the raw accounting data file. Collected data was summarized into forty-eight 30-minute intervals which covered a midnight to midnight

period for each day that data was collected. These intervals represented a 30-minute summary of system activity and were designated as performance summary intervals (PSI). Throughout this report the terms 'PSI' and data set 'case' will be used interchangeably.

The 'awk' report generator and data manipulation utility, and 'sed' stream editor were used to reduce the monitor data, and the 'sa' accounting data summary routine was used to summarize raw accounting data. For the monitor data, reduction primarily involved computing means for monitored activities and writing all means for a given psi to a file. Based on selected options, the 'sa' routine summarized all entries made in the system accounting file for a specified interval. Summarized data included average and per-process cpu utilization; average and per-process memory use (averaged by the amount of cpu time used); and process statistics. After summarization, the three reduced monitor data files and three files created from data extracted from the summarized accounting file were combined into a composite data file. The resulting file contained forty-eight entries or cases for each day on which data was collected. Each case represented one summarized 30-minute period of system activity. As stated above, the cases (PSIs) each contained fifty-two items of data. At the end of this chapter is a breakdown of the data items contained in each case. Data items are numbered (1 through 52) and

named (in paretheses) the way they would appear in the composite data set.

Application of Multivariate Techniques

The overall approach used in applying the techniques involved singly applying each technique to the prepared data set. From a CPE standpoint, the techniques were applied to the data set with the primary goal of revealing dependencies, interdependencies and group-wise relationships among variables in the data set. The types of models which are derived in multivariate analysis are best used in revealing and possibly explaining various types of variable relationships within a multivariate data set. Thus, application of the techniques generally followed a pattern in which 1) objectives were stated for application of the technique, 2) the technique was then applied, 3) statistical considerations were then examined for obtained results and 4) based on statistical significance of the results, the relationships revealed in the derived model were examined for theoretical validity with respect to workload and performance measures represented by the data set variables.

Because the software monitors and accounting data used in constructing the data set provided system oriented rather than job oriented performance data, the prepared data set lent itself well to the type of exploratory

analysis available from the multivariate techniques. Each case in the data set is a 52-variable observation. Thus it would even be difficult for a VAX/UNIX expert to discover, analyze and explain why specific relationships exist among measured variables. However, having exposed relationships in the multivariate data set via use of multivariate analysis techniques, existing relationships and their significance can more easily be identified, and possibly explained.

The following table contains a list of the data items (variables) which made up each case (performance summary interval-PSI) in the prepared data sets. Also given is the column number associated with each of the variables.

Table III-2

Data Set Variable List

columns(1-22) = vmstat data

col #	monitored activity
*case number	cases were numbered from 1 to n to allow easier visual inspection of the data set.
1-casenum	sequence number beginning at 1, given to each case (psi) in the data set.
*case id	label showing day,month and the 30 minute period measured.
2-(per)	
*procs	information about numbers of processes in various states.
3-(r)	number of processes in run queue
4-(b)	number of processes blocked for resources (i/o,paging etc.).
5-(w)	runnable or short sleeper (< 20 secs) but swapped.
*memory	information about virtual and real memory usage.
6-(avm)	active virtual pages
7-(fre)	size of free list
*page	information about page faults and paging activity
8-(re)	page reclaims
9-(at)	
10-(pi)	pages paged in
11-(po)	pages paged out
12-(fr)	pages freed per second
13-(de)	anticipated short term memory shortfall
14-(sr)	pages scanned by clock algorithm,per-second
*disk	information about disk activity
15-(h0)	disk operations per-second for h0
16-(h1)	disk operations per-second for h1
17-(x2)	NO DISK IN USE
18-(x3)	NO DISK IN USE

```
*faults          trap/interrupt rate averages/sec over last
                  five seconds.
  19-(in)         (non clock) device interrupts per-second
  20-(sycl)       system calls per second
  21-(csw)        cpu context switch rate (switches/sec)
 *cpu             breakdown of percentage usage of cpu time
  22-(cpusr)      user time for normal and low priority
                  processes
  23-(cpsys)      system time
  24-(cpidl)      cpu idle

  column (25) = bench mark data
  ------------------------------
  25-(bnchmk)     real time (approx. resp. time) required
                  to execute 'date' command

  column (26) = user process information
  --------------------------------------
  26-(nuprocs)    average number of logged in user processes
                  for 30 min period

  columns (26-31) = disk utilization data
  ---------------------------------------
  filesystem
  27-(rootfs)     % use for logical disk /dev/hp0a
                  (total blks = 7623)
  28-(usrfs)      % use for logical disk /dev/hp0h
                  (total blks = 141545)
  29-(lsfs)       % use for logical disk /dev/hp1g
                  (total blks = 76123)
  30-(enfs)       % use for logical disk /dev/hp1h
                  (total blks = 141578)
  31-(ulfs)       % use for logical disk /dev/hp0e
                  (total blks = 26848)
  32-(tmfs)       % use for logical disk /dev/hp0d
                  (total blks = 7317)

  columns (32-37) = accounting summary data
  -----------------------------------------
  33-(exprocs)        number of executed processes
  34-(realt)          average real time taken for each
                      process
  35-(cpuusac)        average amount of time cpu spent in
                      user mode
  36-(cpsusac)        average amount of time cpu spent in
                      system mode
  37-(avioac)         average number of ios executed
  38-(kac)            average number of memory blocks used
                      over cpu time
```

columns (38-52) = counts for monitored command

39-(troff)	number of processes related to troff and vtroff processing
40-(nroff)	number of processes related ot nroff processing
41-(karel)	number of processes related to karel execution
42-(f77)	number of processes related to fortran compilations
43-(vers)	number of processes related to versatec/ spooler processing
44-(edits)	number of processes related to editor processing
45-(cc)	number of processes related to C compilations
46-(slam)	number of processes related to slam execution
47-(pi)	number of processes related to pascal compilations (interp)
48-(pc)	number of processes related to pascal compilations (compiler)
49-(swmons)	number of processes related to software monitor execution
50-(saproc)	number of processes related to accounting file processing
51-(Srun)	number of processes related to S statistical processing
52-(yaclex)	number of processes related to lexigraph/ compiler processing
53-(utils)	number of processes related to commonly UNIX used utilities

CHAPTER IV

REGRESSION ANALYSIS

Part 1 - Ordinary Least Squares

Analysis Objective

In performing regression analysis on the system performance data, the primary objective was to derive a set of explanative/predictive empirical workload and performance measurement models for the AFIT SSC. Workload parameters of interest include CPU-utilization in both the user (cpusr) and system (cpsys) modes, and average size of the process run queue (runq). The primary performance measure of interest was response time, because it indicated how well the system was handling its current workload. The data set used provided direct measures of cpu utilization and run queue size. However, approximations had to be made for the response time. Two data set items were used to model response time. The first was 'real' execution time (bnchmrk) required for the "date" command, which performs the necessary calculations and table look up to provide the user with the correct day, date, month, time, year and time zone based on a 0000 1 January 1970 epoch. This measure was used to indicate response time for a given period of

system activity. The second approximation for response time was taken from the system accounting files. Among the accounting data is average 'real' execution time taken by all processes run over a given time period (realt). This measure is not as good as single-command measure of response time because over a given interval the processes executed could vary extensively in the amount of time taken for execution. Thus, this measure only gives a very general indication of what response time may have been and is dependent on the types of processes being run for the measured interval.

After the models were established, the cpu-user mode utilization model was further examined to see if it was the same for each day in the three day period covered. To do this a general linear test (GLT) was performed. The composite data matrix was split into matrices that represented each of the three days. Regressions were then performed using each days data to obtain the anova results necessary to compute the full model SSE. When performing a GLT, the reduced model is the one built using the entire data matrix. Then using the assumption that,

$$"SSE_{full} = SSE_1 + SSE_2 + SSE_3" \quad (7)$$

and

$$"df_{full} = df_1 + df_2 + df_3" \quad (8)$$

the F statistic was computed using,

$$F = \frac{\dfrac{SSE_{reduced} - SSE_{full}}{df_{reduced} - df_{full}}}{SSE_{full} / df_{full}} \qquad (9)$$

Analysis Tools

This part of the regression analysis was performed using the 'S' data analysis/manipulation package available on the AFIT SSC. S provided many lower level data manipulation capabilities not available in SPSS or BMDP. (Ref 22).

Analysis Results

The first portion of the regression analysis project entailed using the S 'cor' function to produce a 51x51 correlation matrix of all variables used in the data set. The matrix was then examined to determine inter-variable correlations. Based on this visual inspection of the data, independent variables which had correlations of $r \geq .6$ with selected dependent variables were chosen.

Using the regression by 'leaps and bounds' technique available in S, all-way regressions were run for each of the possible subset sizes 1 through n, where n is the number of independent variables used in the regression. The results of the 'leaps' procedure provides either the Cp criterion, R-square, or the adjusted R-square as model

comparison measures. Using the Cp statistic which is concerned with the total squared error of the n fitted observations for any given model, four models were selected from the leaps results for each of the dependent variables being examined. For the dependent variables being modeled, the models rendering the lowest Cp statistic were:

(explanations of variable names are given in Chapter III)

$$cpusr = 204.6433 - 0.0243fre - 5.0979pi + .9099h0 + 2.5776hl - 1.56847usrfs$$

$$cpsys = 3.7825 - (0.1238e-2)fre + 2.7304pi - 2.3511po - 3.1971fr + 5.5214de - 0.1506sr + 0.1611h0 + 0.3983hl + 0.1022in - (0.1045e-2)sycl + (0.4808e-1)csw + (0.4804e-1)nups + (0.3229e-2)exps + (0.1917e-1)f77$$

$$bnchmrk = 1.0381e2 - (3.3162e-4)avm - (2.0474e-3)fre + (9.5810e-2)hl - (9.9347e-1)cpusr - 1.0383cpsys$$

$$realt = 47.4816 - 0.7507r - (0.8413e-3)avm + 1.8493re - (9.4303)po + 2.5419fr + 1.354troff$$

$$runq = -.4747 - 42.4565w + (0.2749e-2)avm - 2.6134pi + 0.4643sr + .3258h0 - 0.0937in + 0.0522csw$$

Regression results are shown in Table IV-1 at the end of part 1 of this chapter. Each of the selected models were examined for violations of regression model assumptions. This was done using residual analysis and examination of the coefficient correlation matrix for each regression. Residual normality(N), heteroscedascticity(H) and autocorrelation(A) were checked by using plots of the error terms versus 0, y, and time. The plots revealed the following for each derived model:

cpusr model

N --> VALUES EVENLY DISTRIBUTED ABOUT 0

H --> EVIDENCE OF POSITIVE CORRELATION WITH Y VALUES

A --> NO CYCLIC PATTERNS OVER TIME

cpsys model

N --> VALUES EVENLY DISTRIBUTED ABOUT 0

H --> NO VISUAL CORRELATION WITH Y-HAT VALUES

A --> NO CYCLIC PATTERNS OVER TIME

bnchmrk model

N --> APPEARS TO HAVE BIMODAL DISTRIBUTION ABOUT ZERO

H --> SHOWS EXPANDING FUNNEL PATTERN WITH INCREASING Y VALUES

A --> POSSIBLE CYCLIC PATTERNS AT 0,50,100 INTERVALS

realt model

N --> EVENLY DISTRIBUTED ABOUT 0, POSSIBLE OUTLIER

H --> STRONG POSITIVE CORRELATION WITH Y VALUES

A --> NO CYCLIC PATTERNS OVER TIME

runq model

N --> EVENLY DISTRIBUTED ABOUT 0

H --> SLIGHT POSITIVE CORRELATION WITH Y VALUES

A --> NO CYCLIC PATTERN OVER TIME

Examination of the coefficient correlation matrices revealed the following multicollinearity problems among the independent variables:

cpusr model: .6817 correlation between 'fre' and 'hl'.

cpsys model: .5424 correlation between 'in' and 'fre'
-.6538 correlation between 'fr' and 'pi'
-.6785 correlation between 'fr' and 'po'
-.5077 correlation between 'in' and 'h0'
-.6891 correlation between 'h0' and 'csw'
-.6051 correlation between 'in' and 'hl'

bnchrmk model: -.5944 correlation between 'fre' and 'hl'
.5605 correlation between cpsys' and 'fre'

realt model: -.5619 correlation between 'fr' and 're'
-.8021 correlation between 'fr' and 'troff'

runq model: -.5492 correlation between 'avm' and 'sr'
-.5563 correlation between 'sr' and 'pi'

As shown above only correlations above .500 were identified as being problems. While the pair wise comparison rendered the above results for single order interaction, no checks were made for higher order interactions. Higher order interactions, which could also exist, are those in which combinations of two or more independent variables are correlated with one or more of the other independent variables. The cases of multicollinearity that do exist all convey the actual

relationships of the independent variables involved. For example, the correlations between the fr, sr, re, sr, pi, and po variables exist because these variables all represent various forms of paging activity.

To test whether or not the model developed for cpusr' applied to each of the 3 days for which data were collected, a general linear test was performed. Results of the test revealed the following:

Null Hypothesis: Corresponding coefficients for each independent variable in the models for each day are equal.

Alternate Hypothesis: At least one set of corresponding coefficients are not equal.

Using the Eq 9, the calculated F statistic was 3.98 with 6 and 125 degrees of freedom. With the degrees of freedom given the critical value is 2.10. Thus, the null hypothesis can not be accepted and it must be concluded that at least 2 of the models from the 3 day period are significantly different (at α = .05) from the composite model.

Analysis of Results

Results of the regression analysis rendered models that all presented an intuitive feel for how various system activities were related to each other. For example, the 'runq' model indicates that size of the process run queue is dependent upon the number of swapped processes, amount of memory being used, paging and disk activity, and the number of device interrupts being processed by the cpu. Thus, based on the signs of the coefficients one would expect the process run queue to increase in size as memory usage and disk activity increased and as, swapped processes, paging activity, and device interrupts decreased.

However, because of the problems that were revealed in the residual analysis, all of the models must be further developed before they can be used for reliable interpretation. For example, in the models for 'bnchmrk' and 'runq,' adding dummy variables to represent each of the 3 days or times of day (i.e. 0600-1200, 1200-1800, etc.) could provide a possible solution to the apparent autocorrelation problem.

Further Analysis

Regression models for the same dependent variables

were also derived using SPSS. In addition to those models developed using S, active virtual memory use (avm) was also modeled using SPSS. While the results were not exactly the same the models developed were similar. The differences appeared to be due to the fact that the step-wise variable selection procedure used by SPSS and the regression by leaps and bounds technique used by S used different heuristics to decide which variable(s) to include next in the model. Thus, some of the SPSS models consisted of different variables. However, all highly significant variables were present in models developed by both systems.

An attempt was made to perform a GLT using the SPSS output, however, after eight significant variables in the full data set had been chosen for examination, it was discovered that four of them displayed extreme changes in significance in data sets representing each day. Thus, the GLT could not be performed with those variables selected. The changes in the day to day significance were used as basis for concluding that the models developed for the full data set were significantly different from those developed for each one day period. This conclusion provided further indication that the performance and workload measures modeled were time dependent at least over a weekly period. The changes in significance could also be attributed in part to the effects of multicollinearity on computed regression coefficients.

REGRESSION RESULTS FROM S PROGRAM

Table IV-1a

Cpu user-mode utilization model

A N O V A

SSR= 85625 dfR= 5 MSR= 17125

SSE= 26224.94 dfE= 137 MSE= 191.423

VAR	COEF	STD ERR	T-VAL
int	204.643	33.272	6.151
fre	-0.024	0.003	-7.238
pi	-5.098	1.113	-4.582
h0	0.909	0.314	2.894
h1	2.578	0.445	5.788
usrfs	-1.568	0.361	-4.340

Residual Standard Error = 13.856

Multiple R-Square = 0.766

N = 143

F = 89.46 on 5, 137 df

Table IV-1b

Cpu system-mode utilization model

A N O V A

SSE= 253.464 dfE= 128 MSE= 1.98

VAR	COEF	STD ERR	T-VAL
int	3.783	1.427	2.650
fre	-0.001	0.001	-2.375
pi	2.730	0.394	6.922
po	2.351	0.582	4.036
fr	-3.179	0.539	-5.893
de	5.521	1.384	3.989
sr	-0.150	0.048	-3.163
h0	0.161	0.064	2.494
h1	0.398	0.057	6.980
in	0.102	0.017	5.948
sycl	-0.001	0.001	-1.674
csw	0.048	0.006	7.246
nups	0.048	0.013	3.689
exps	0.003	0.001	5.071
f77	0.019	0.014	1.347

Residual Standard Error = 1.407 N = 143

Multiple R-Square = 0.988 F = 730.91 on 14,128 df

Table IV-1c

Benchmark Response Time Model

A N O V A

SSE= 97.4 dfE= 137 MSE= 0.71115

VAR	COEF	STD ERR	T-VAL
int	1.04e2	0.851	122.0
avm	-3.31e-4	0.001	-3.406
fre	-2.04e-3	0.001	-6.518
hl	9.58e-2	0.033	2.912
cpusr	-9.93e-1	0.005	-215.418
cpsys	-1.03e0	0.018	-58.252

Residual Standard Error = 0.8433

Multiple R-Square = 0.999

N = 143

F = 56258.23 on 5, 137 df

Table IV-1d

Average Real Time Response Time Model

A N O V A

SSE= 20434.07 dfE= 136 MSE= 150.251

VAR	COEF	STD ERR	T-VAL
int	47.482	1.708	27.806
r	0.751	0.324	2.316
avm	-0.001	0.001	-0.461
re	1.849	0.703	2.632
po	-9.430	4.619	-2.042
fr	2.542	2.881	0.882
troff	1.354	0.479	2.823

Multiple R-Square = 0.326

N = 143

F = 10.94 on 6, 136 df

Table IV-1e

Process Run Queue Model

A N O V A

SSE= 398.87 dfE= 135 MSE= 2.955

VAR	COEF	STD ERR	T-VAL
int	-0.475	0.264	-1.795
w	-42.457	25.071	-1.693
avm	0.003	0.001	10.117
pi	-2.613	0.245	-10.678
sr	0.464	0.040	11.557
h0	0.326	0.076	4.277
in	-0.094	0.014	-6.470
csw	0.052	0.006	8.442

Residual Standard Error = 1.7189

Multiple R-Square = 0.948

N = 143

F = 352.2 on 7, 135 df

Part 2 - Ridge Regression

Analysis Objective

As was evident in the application of ordinary least squares regression in part one of this chapter, OLS regression modeling can be adversely affected by violations of required assumptions. One of those assumptions is that the independent variables are not correlated with each other. Violation of this assumption results in multicollinearity and in extreme cases will render an ill conditioned **X'X** matrix which cannot be inverted. Even when multicollinearity is not extreme ($r \geq .95$), its presence at moderate levels ($.4 \leq r \leq .7$) will result in unstable regression coefficients that do not accurately reflect the independent variables relationship to the dependent variable. A technique which attempts to adjust for multicollinearity is ridge regression.

To examine the use of ridge regression, variables selected by SPSS for the ordinary least squares (OLS) regression model for active virtual memory use (avm) were used. The ridge program used in this analyses limited the total number of variables which could be used to sixteen. The regression algorithm used in the ridge program also did not allow for step wise inclusion of independent variables. Instead, all variables available were included and the computed regression coefficients were based on the initial

value of the $(X'X)^{-1}$ data matrix. Thus, the regression coefficients computed were not exactly the same as those computed by S or SPSS. As a result, the ridge models developed could not be directly compared to those developed using the other analysis tools. However, the ridge program provided values for regression coefficients computed without the addition of bias which provided its equivalent to the OLS coefficients. These OLS coefficients were used as the basis of comparison between models.

The first objective was to examine the ridge trace generated by the ridge program. This would provide the necessary indication of how much instability existed in the repression coefficients. Based on the ridge trace the decision could be made about the accuracy of the OLS model. The second objective was to then examine the computed ridge coefficients and determine which set of them would improve the regression model. The selection of ridge coefficients was based on the ridge trace and two heuristics which had been used in past studies to select the best set of coefficients.

The ridge program was run using a bias increment value of .005 (k=.005). The program computed regression coefficients for all values of k between 0 and .245. Also provided were the correlation matrix, the ridge trace and the Variance for Inflation Factors, (VIFS). The OLS model coefficients were:

+34.3r	-155.0b	-12500.0w	+175.0fr
-4.81cpidl	+30.5nuprocs	-21.0usrfs	-53.1enfs
-43.0ulfs	-36.8troff	+4.05edits	+41.3cc
+3.32pc	-32.4srun	+2.09utils	

const = 16280.0

Examination of the correlation matrix, the ridge trace and the VIFs revealed that these coefficients were highly unstable due to the effects of multicollinearity betwen independent variables. The independent variables exhibiting significant multicollinearity ($r \geq .5$) were:

r	fr	troff
b	cpidl	edits
w	usrfs	utils

The ridge trace generated by the ridge program is shown in Figure IV-2. The trace shows how each of the coefficients varied with increasing values of k. The trace revealed that coefficients for nuprocs(7), r(1), and w(3) were the most unstable. While nuprocs and r showed high rates of change in coefficient value, w's coefficient eventually changed signs when a certain level of bias had been reach. Coefficients which swing from negative to positive usually do so because of the effects of multicollinearity.

```
RIDGE TRACE: NORMALIZED COEFFICIENTS
COEFFICIENT RANGE:      -.1299 TO       .6183
K-VALUE ...................................................... R-SQUARE

 .000    63 BBF . ED CG   15                                   7  .9869
 .005     6 3 BF . ED C G   51                              7     .9868
 .010   6   23BF . E D CG   5 1                          7        .9865
 .015   6   2 BF . E D CG   5  1                      7           .9862
 .020   6 2   9F . E C  G   5  1                   7              .9859
 .025   6 2   9F . E D  G   5   1                7                .9856
 .030   6 2   9F . E D  G   5   1              7                  .9852
 .035   6 2   9F3. E D  G   5   1            7                    .9849
 .040   6 2   AF3. E C  G   5   1          7                      .9846
 .045   6 2   AF3. E D G C  5    1         7                      .9842
 .050   6 2   AFB3 ED   G C 5    1       7                        .9839
 .055   6 2   AFB3 ED   G C 5    1      7                         .9835
 .060   6 2   AFB3 ED   G C 5    1      7                         .9832
 .065   6  2 AFB.3ED    G C 5    1     7                          .9828
 .070   6  2 AFB.3ED    G C 5    1     7                          .9825
 .075   6  2 A F.3ED    G C 5    1     7                          .9821
 .080   6  2 A F.3ED    G C 5    1    7                           .9818
 .085   6  2 A F.3ED    G C 5    1    7                           .9814
 .090   6  2 A F. ED    G C 5    1    7                           .9811
 .095   6  2 A FB 3E    G C 5    1   7                            .9807
 .100   6  2 A FB 3E    G C 5   1    7                            .9804
 .105   6  2 A FB 3E    G C 5   1    7                            .9800
 .110   6  2 A FB 3E    G   C5  1   7                             .9797
 .115   6  2 A FB  E    G   C5  1   7                             .9794
 .120   6   2A FB  E    G   C5  1   7                             .9790
 .125   6   2A FB  E    G   C5  1   7                             .9787
 .130   6   2A FB  E    G   C5  1  7                              .9783
 .135   6   2A FB  E    G   C5  1  7                              .9780
 .140   6   2A FB  E    G   C5  1  7                              .9777
 .145   6   2A FB  E    G   C5  1  7                              .9773
 .150   6   2A FB  E    G   C5  1  7                              .9770
 .155   6   2A FB  E3   G   C5  1  7                              .9767
 .160   6   2A FB  E3   G   C5  1 7                               .9763
 .165   6   2A FB  E3   G   C5  1 7                               .9760
 .170   6   A98FB  E3   G   C5  1 7                               .9757
 .175   6   A98F.B E3   G   C5  1 7                               .9753
 .180   6   A98F.B E3   G   C5  1 7                               .9750
 .185   6   A98F.B E3   G   C5 1  7                               .9747
 .190   6   A98F.B E3   G   C5 1 7                                .9744
 .195   6   A98F.B E3   G   C5 1 7                                .9741
 .200   6   A98F.B E3   G   C5 1 7                                .9737
 .205   6   A98F.B E3   G   C5 1 7                                .9734
 .210   6   A98F.B E3   G   C5 1 7                                .9731
 .215   6   A98F.B E 3G     C5 1 7                                .9728
 .220   6   A98F.B E 3G     C5 1 7                                .9725
 .225   6   A98F.B E 3G     C5 1 7                                .9722
 .230   6   A98F.B E 3G     C5 1 7                                .9719
 .235   6   A98F.B E 3G     C5 1 7                                .9716
 .240   6   A98F.B E 3G     C5 1 7                                .9712
 .245   6   A98F.B E 3G     C5 1 7                                .9709
```

Coefficients codes

1- r
2- b
3- w
4- avm (dep)
5- fr
6- cpidl
7- nuprocs
8- usrfs
9- enfs
10- ulfs
11- troff
12- edits
13- cc
14- pc
15- srun
16- utils

Examination of the computed VIFs for the OLS model showed that four of the variables (r, nuprocs, edits, utils) had VIFs greater than 10, which indicates that these variables were correlated with at least one other independent variable at $r \geq .9$. Thus, based on these observations, it was concluded that the explanatory value of the OLS model for active virtual memory use was limited.

Since .005 was used as the increment value for k, 50 sets of regression coefficients were computed. Selection of which set would provide the best model can not be based on any fixed rule or algorithm. Rather, use of the ridge trace, or selection heuristic or both must be incorporated to select a coefficient subset. Use of the ridge trace is the most common selection technique. This entails selecting a value of k where all or most of the regression coefficients have stabilized. Based on this method of selection the following model was selected with bias being k = .130:

avm = -5054 + 51.4r - 166b + 5770w + 152fr
- 5.61cpidl + 15.7nuprocs
- 18usrfs - 81enfs - 63.7ulfs
- 1.4troff + 7.57edits + 40.7cc
+ 3.93pc - 18.7srun + 2.03utils

Using the heuristic, all VIFs less than 10, a second model was selected. This model, required a bias of k = .110:

Model based on -- All VIFs less than 10

avm = -3624 + 52.2r - 179b + 4580w + 154fr
- 5.68cpidl + 16.4nuprocs
- 18.3usrfs - 82.4enfs - 61.3ulfs
- 3.90troff - 7.46edits + 41.0cc
+ 3.82pc - 20.4srun + 2.05utils

A final model was selected using the heuristic -- all signs correct. Coefficients were chosen at the k value where all negative coefficients, which intuitively should have been positive, had transitioned from negative to positive. The variables which had coefficients that changed sign were w and troff. The model selected using signs-correct heuristic required a bias of k = .145:

Model based on -- All signs correct

avm = -5961 + 50.8r - 156b + 6530w + 151fr
- 5.55cpidl + 15.3nuprocs
- 17.8usrfs - 79.9enfs - 65.3ulfs
+ .272troff - 7.63edits + 40.4cc
+ 4.01pc - 17.5srun + 2.02utils

Examination of the unnormalized regression coefficients in the models presented showed how they varied with increased bias. However, it was necessary to examine the normalized coefficients to determine how the explanatory power of the model was effected by the bias. Based on the OLS normalized coefficients, active virtual memory use was most strongly affected by the number of user

processes that were currently running on the system. With a normalized coefficient of .618, interpretation using the OLS model would identify 'nuprocs' as the most significant variable in determining 'avm.' However, the normalized ridge coefficients at k = .145, showed that the size of the process run queue and the number of editor processes were significant variables in determining 'avm.'

At the .145 bias level, the coefficient for 'nuprocs' had dropped to .310 while the coefficient for 'runq' had increased from .149 to .221, and 'edits' had increased from .068 to .127. These changes in coefficient size are made evident in the ridge trace, which graphically depicts the effect of increasing bias on the size of the coefficients. Thus, as anticipated, use of ridge provided more stable coefficients and consequently enabled more accurate explanations of the relative importance of the independent variables.

To evaluate the effect of the added bias on the predictivity and significance of the models, an S program was developed to compute the SSE for each model. The SSE was then used along with the R^2 value computed in the ridge program to compute an overall F value for the model.

Table IV-1 shows R^2 versus k for each of the developed models. As shown the R^2 value decreased as bias was increased, however, the computed overall F-statistic obtained from the S program output remained significant at α = .05 for each model. Thus, the injected bias resulted in a slight decrease in the R^2 for the model but the model still remained significant. The model selected using the heuristic -- all signs correct, had the highest bias but still had an R^2 of .9773, a decrease of .0096 from the OLS model R^2. The small decrease in R^2 indicates that the predictivity of the model was not adversely affected by the added bias.

Table IV-2

Comparison of 'avm' models

Model	Bias	R^2
OLS	0.0	.9869
VIFs < 10	0.110	.9797
Trace	0.130	.9783
Signs	0.145	.9773

Conclusion

The OLS regressions performed in this chapter reveal the value of this data analysis technique as a modeling tool to the CPE analyst. However, when regression is used for explanative applications, the data used to build the

models and the resulting residuals must meet the required independence and normality assumptions. Ridge regression provides an alternative to the OLS regression which effectively negates the effects of multicollinearity. Though the ridge models had biased coefficients, the overall significance and the predictive capability of the models were not greatly reduced. Thus the CPE analyst can use ridge regression to improve the explanatory power of models which are developed from data exhibiting multicollinearity. Because computer system performance data, in many cases, exhibits strong inter-relationships, ridge regression would be a useful tool if the CPE analyst wanted to use the data to build explanatory models.

CHAPTER V

CANONICAL CORRELATION ANALYSIS

Analysis Objective

In some data analysis situations it may be more advantageous to examine relationships between groups of variables. The question posed would be whether or not a significant relationship exists between two sets of variables. Based on the existence or non-existence of this relationship, hypotheses can be formed about why the relationship does or does not exist. If a relationship does exist then further hypotheses can be made about the strength of the relationship. Though some of these hypotheses may not be statistically testable, they can provide the CPE analyst with insight into the underlying structure of collected CPE data and consequently may lead to revelations of causal relationships between system activities. Canonical correlation provides the analyst with a means to examine the strength of relationships between groups of variables.

Visual examination of the collected CPE data reveals that a number of the variables are related to a common system activity or function. As a result it should be possible to linearly combine the related variables in such

a way that they can be expressed with the linear combination alone. If this was the only goal of the analysis, then factor analysis could be used. However, an additional requirement for the analysis is to build the linear composites so that the super variable created will be maximally correlated with another super variable created using the same linear combination technique. Thus, canonical correlation will be used to accomplish the analysis.

CPE data variables contained in the dataset can be grouped by system function or activity into eight categories:

1) Variable Set:Process Execution Status

variables = R, B, W

2) Variable Set:Memory Activity

variables = AVM, FRE, KAC, RE, PIN, PO, FR, DE, SR

3) Variable Set:Disk Activity

variables = H0, H1, AVIOAC

4) Variable Set:Processor Interrupt Activity

variables = INT, SYCL, CSW

5) Variable Set:Cpu Utilization Status

variables = CPSYS,CPUSR,CPIDL,CPUUSAC,CPUSYAC

6) Variable Set:Disk Utilization

variables = ROOTFS, USRFS, LSFS, ENFS, U1FS, TMFS

7) Variable Set:Process Mix

variables = TROFF, NROFF, VERS, EDITS, CC, F77, PI
PC, KAREL, SWMONS, SAPROC, UTILS, SLAM,
SRUN, YACLEX

8) Variable Set:Response Time Indicators

variables = BNCHMK, REALT

As can be seen from the variable sets described, further decomposition of the sets is possible for super variables like memory activity and process mix. However, for this analysis only the variable sets listed were used.

Canonical correlation was used to establish how each of three of the variable sets (1-process execution status, 5-cpu utilization, 8-response time indicators) relate to the remaining seven. To accomplish this part of the analysis, only the canonical correlation as computed by the SPSS subprogram CANCORR, was used. Of particular interest were relationships that existed for the cpu utilizaton group of variables. For this variable set the analysis was taken further and canonical loadings were computed using the file modification capability of SPSS. Using these loadings, the derived super variables will be analyzed for possible interpretation. Finally, the constituent variables for the created super variables were examined for information overlap. This was accomplished using Stewart and Love's technique for measuring redundancy.

Once the canonical correlation analysis was complete, the relationships revealed between the super variables were used to describe system activity at the general level (i.e. set apart from the detailed activity as described by the individual variables).

Analysis Results

The first variable set to be analyzed was the 'response time indicators.' Table V-1 shows the significant canonical correlations for response time indicators and the remaining seven super variables as computed by SPSS. Though no predictor/criterion variable relationship really exists in a canonical correlation, in this analysis the super variable being compared to the remainder of the super variables will be designated as the criterion variable for model naming purposes.

Table V-1

Canonical Correlations for

Response Time Indicators

CRITERION VARIABLE: Response Time Indicators

PREDICTOR VARIABLES	CANONICAL CORRELATION
Cpu Utilization(1)	.72141
Cpu Utilization(2)	.31150
Memory Activity(1)	.78443
Memory Activity(2)	.44866

Table V-1

Canonical Correlations for

Response Time Indicators (cont'd)

Disk Activity/Disk Utilization	.74942
Processor Interrupt Activity	.69591
Process Status(1)	.72656
Process Status(2)	.26945
Process Mix(1)	.76651
Process Mix(2)	.57807

As shown in Table V-1, 'response time indicators' are significantly correlated to each of the remaining super variables with .72 correlation or greater for initial canonical variates. Of the super variables shown, 'memory activity(1)' has the highest correlation and explains 61.5% of the variation in 'response time indicators.' This indicates that increases in memory activity (i.e page swapping) in the processing environment will result in some increase in response time indicators.

Computed canonical correlation for 'process execution status' and the seven remaining super variables are shown in Table V-2.

Table V-2

Canonical Correlations for Process Execution Status

CRITERION VARIABLE: Process Execution Status

PREDICTOR VARIABLES:	CANONICAL CORRELATION
Cpu Utilization Status	.84810
Memory Activity(1)	.97231
Memory Activity(2)	.62285
Memory Activity(3)	.49167
Disk Activity and Utilization(1)	.91215
Disk Activity and Utilization(2)	.74038
Disk Activity and Utilization(3)	.38088
Process Mix(1)	.75124
Process Mix(2)	.64355
Processor Interrupt Activity(1)	.95907
Processor Interrupt Activity(2)	.54153

With 'process execution status' the correlations for the first canonical variates are higher, with four of the six shown being above .80. Examination of the corresponding eigenvalues shows that variation in 'process execution status' is over 90% explained by each of the super variables 'memory activity' and 'processor interrupt activity'. Thus, increases either of the two super variables will cause the process execution status to increase.

The canonical correlations for the final variable set analyzed are shown in Table V-3. Unlike the analysis for the forementioned super variables, 'cpu utilization' was further analyzed for interpretation and information overlap.

Table V-3

Canonical Correlations for Cpu Utilization

CRITERION VARIABLE: Cpu Utilization Status

PREDICTOR VARIABLES:	CANONICAL CORRELATION
Cpu Utilization Status(1)	.84602
Memory Activity(1)	.95047
Memory Activity(2)	.53737
Disk Activity(1)	.96621
Disk Activity(2)	.48634
Process Mix(1)	.92100
Disk Utilization(1)	.70037
Disk Utilization(2)	.63601
Processor Interrupt Activity(1)	.97951

With the exception of 'disk utilization' , 'cpu utilization status' was correlated at .92 or greater with each of the other super variables. The corresponding eigenvalues indicated that the given super variables could explain at least 90% of the variation in 'cpu utilization status.'

Extended Analysis of Cpu Utilization

One of the performance measures of interest in the group of super variables used in the canonical correlation analysis was 'cpu utilization status'. The variable set which made up this super variable was further analyzed for information overlap with each of the other canonical variate variable sets. This analysis was performed using Stewart and Love's procedures for calculating an overall redundancy measure. The following formulas were used in the calculations:

$$v_{y^*} = \sum \frac{r^2_{y^*yj}}{p} \qquad (10)$$

where v_{y^*} is the proportion of criterion variance explained by y^* and r^2 is the squared loading of y_j on y^*.

$$R^2_{y^*} = \sum (v_{y^*})(r^2_{y^*x^*}) \qquad (11)$$

where R^2 is the total redundancy in the criterion set given the predictor set and the summation is across all canonical correlations (Ref 14:5-41 - 5-43).

The redundancy specifies the fraction of the variance in the y's which can be explained by an optimal linear combination of the x's, just as the multiple regression

analysis R^2 specifies the fraction of variance in a single y which can be explained by an optimal linear combination of x's (Ref 14:5-12). Prior to calculation of the redundancy measures, canonical loadings had to be computed for each of the constituent variable set. The canonical loadings reveal which of the constituent variables load most heavily with the canonical variate and hence which variables are most important in super variable interpretation. Table V-4 shows the computed redundancy measures for the 'cpu utilization status' given each of the predictor variable sets.

Table V-4

Redundancy Measures for Cpu Utilization Status

CRITERION VAR	PREDICTOR VAR	REDUNDANCY
cpu utilization	process status	.5416
cpu utilization	memory activity	.7328
cpu utilization	disk activity	.7946
cpu utilization	disk utilization	.4623
cpu utilization	process mix	.6935
cpu utilization	processor intrpt activity	.7892

The redundancy measures in Table V-4 revealed that the predictor variable sets for 'memory activity' 'disk activity' and 'processor interrupt activity' each account for over 70% of the variation in 'cpu utilization status'.

Interpretative Analysis

Examination of the canonical correlations for 'cpu utilization status' revealed that one of the strongest super variable relationships existed with 'processor interrupt activity'. Variables included in this set were 'int' (device interrupts per second), 'sycl' (system calls per second) and 'csw' (context switch rate). Using the cpu utilization canonical variate as the dependent variable and processor interrupt activity as the independent variable, the redundancy measure in Table V-3 indicates that 78.92 per cent of the variation in the cpu utilization status can be explained by the interrupt activity.

Using the canonical loadings to determine which of the substituent variables the canonical variates most represent, implications of the computed redundancy can be examined. Canonical loadings, computed using the SPSS Pearson Corr procedure, revealed that the cpu utilization variate has a strong negative correlation with 'cpsys' ($r = -.9999$) while the cpu interrupt variate has a strong negative correlation with 'int' ($r = -.9955$). Thus, as 'int' increases, the cpu interrupt variate decreases and a corresponding decrease will take place in the cpu utilization variate. Since the cpu utilization variate is negatively correlated with 'cpsys,' 'cpsys' would be expected to show a corresponding increase. The strong correlation between these two canonical variates indicates

that processing conditions which contain extensive processor interrupt activity will result in high cpu system-mode utilization.

Other strong relationships that exist reveal that increases in memory and disk activity (which are themselves related) will also coincide with or result in increased cpu utilization. In all of the relationships examined, the computed correlations and redundancy measures were intuitively appealing. As a result, the canonical correlation analysis provided no grounds to suspect abnormal system behavior with respect to the variable groups that were analyzed. If one of the examined relationships had exhibited an unusual system behavior such as decreasing cpu utilization as interrupt activity increased, then possible system malfunctions may exist which warrant further examination. Herein, lies the primary usefulness of canonical correlation, the ability to examine the nature of multivariate relationships within performance data sets.

Conclusions

The canonical correlation performed in this chapter revealed that it is a useful means of examining correlations between groups of variables. In multivariate performance data sets, the ability to examine how one group of parameters varies with a second group of measures, can

CHAPTER VI

FACTOR ANALYSIS

Analysis Objective

An analyst faced with a large set of performance and workload data may want to determine how much of the data contains useful information. In many cases a large set of CPE data will contain redundant information due to different monitors having measured the same system or job related activity. Factor analysis provides the analyst with a means of examining the dimensionality of a data set and possibly reducing the dimensionality if redundant information is present.

The primary objective of this analysis is to explore the interdependencies that exist among the 52 variables that make up the CPE data set. It is hoped that the size of the data set can be reduced to a small number of factors which contain the same information about system performance.

For this analysis, principal component factoring without iterations was used first. After completing the principal component analysis, a second factor analysis was done using classical factor analysis. With the principal component technique all three of the rotation techniques

available in SPSS were used in the interpretation phase. However, with classical factor analysis only the VARIMAX rotation was used.

Since factor analysis is concerned solely with interdependencies in the data, it was not necessary to select dependent variables from the data set. For the principal component analysis, the CPE data set was entered to SPSS in its entirety, without variable modification or exclusion. However, with classical factor analysis four of the variables which were measured at zero for the full monitored period were excluded so that the determinant could be computed. The variables that were omitted consisted of 2 which represent disk drives that are not currently installed on the system and two that represent monitored processes that were not executed during the data collection period. Other than this modification, no other modifications were made outside of SPSS.

Part 1 - Principal Component Analysis

Analysis Results

In performing principal component factoring, the first part of the analysis involved the entire CPE data set. Fifty one factors were computed with the first forty one accounting for 100% of the variation in variables of the data set. Of those forty one factors, only the first nine would be kept using the factor retention rule of thumb,

$\lambda > 1$. As an additional retention test, a scree test was performed and also resulted in the retention of the first nine factors. Bartlett's Sphericity test was not performed due to the size of k.

The nine retained factors account for 79.1% of the variance in the CPE data set. Table VI-1 shows a break out of the factors and corresponding eigenvalues, with the individual and cumulative percents of variation explained.

Table VI-1

Selected Principal Component Factors

FACTOR	EIGENVALUE	PCT	CUM PCT
1	19.463	41.4	41.4
2	4.895	10.4	51.8
3	3.583	7.6	59.4
4	2.116	4.5	63.9
5	1.977	4.2	68.2
6	1.493	3.2	71.3
7	1.377	2.9	74.3
8	1.181	2.5	76.8
9	1.085	2.3	79.1

As shown in the Table VI-1, factor 1 (F1) accounts for 41.4% of the variance alone. While this is over half of the total variance accounted for by the retained factors, F1 does not account for enough of the variance in the entire data set to allow consideration of the underlying process

as being one dimensional. As expected, examination of the factor matrix showed that the majority (29) of the 51 variables loaded F1 with loadings of .5 or greater. Since this high loading complicated interpretation, the VARIMAX rotated factor matrix was the first rotated factor structure used to simplify interpretation.

Using .5 or greater as an arbitrary criterion for significant loadings, the unrotated factor matrix shows that two of the variables (hl,at) had high loadings on two of the retained factors. Thus, two variables in the data set have a complexity of two.

Initially, only the VARIMAX rotated factor matrix was used for interpretation. While the rotation did give a more even distribution of factor loadings, four variables had a complexity of two (b,avm,cpusr,nuprocs). In spite of the possible complexity problem, interpretation was attempted using variable/factor groupings shown in Table VI-2.

Table VI-2

Principal Component Factor Groupings

FACTOR 1		FACTOR 2		FACTOR 3	FACTOR 4
FRE	TROFF	R	FR	H1	B
AT	NROFF	W	ROOTFS	CPUSYAC	SYCL
H0	F77	AVM	REALT	CPUUSAC	CSW
INT	VERS	RE		AVIOAC	
CPUSR	EDITS	PIN		KAC	
CPSYS	SWMONS	PO		ENFS	
CPIDL	UTILS	SR		USRFS	
EXPS	LSFS	NUPROCS			
PC	BNCHMK				

FACTOR 5	FACTOR 6	FACTOR 7	FACTOR 8	FACTOR 9
U1FS	CC	DE	KAREL	SAPROC
TMFS				
PI				

Resulting interpretation of retained factors is shown in Table VI-3.

Table VI-3

Factor Interpretations

Factor 1 =	Processing Flow Intensity
Factor 2 =	Processing Flow Resistance
Factor 3 =	Accounting Measures of Processing Activity
Factor 4 =	Processing Interrupt Activity
Factor 5 =	UlFS/TMFS File System Utilization
Factor 6 =	C/Pascal Compiler processing
Factor 7 =	Short Term Memory Shortfall
Factor 8 =	Karel Processing
Factor 9 =	Accounting File Processing

As shown in Tables VI-2 and VI-3 more variables load F1 than any other factor and as a result the interpretation is the most general. To accomplish additional alternative interpretations EQUIMAX and QUARTIMAX rotations were performed. Interpretations using QUARTIMAX rotation required an even more general description of factor 1 because variables which loaded factor 2 with VARIMAX rotation now tended to load factor 1. Also, seven variables now had a complexity of 2 with loadings of .5 or greater. EQUIMAX rotation provided even distribution of variable loadings on the retained factors. However, the combination of variables that resulted could not be easily given theoretical meaning. Thus, because of the problems which

arose with QUARTIMAX and EQUIMAX rotations, only the interpretation using VARIMAX will be used.

Extended Analysis Using Computed Factors

A second phase of principal component analysis was also to produce a regression model using the retained orthogonal factors as independent variables. Of particular interest from a CPE stand point is the model of system response time to user interactive commands. Since variable 'bnchmk' provides the best indication of this performance measure, a second principal component analysis was performed using all variables except 'bnchmk'. As expected, results from the analysis were very similar to those for the analysis of the entire data set. While most of the loadings showed a slight increase, the same variables loaded the same rotated factors, rendering the same factor interpretations.

Using the FACSCORE parameter in the SPSS FACTOR procedure, factor scores were computed for each case in the CPE data set. SPSS file modification techniques were then used to add the computed scores to the data set as variables F1 through F9. Finally, a regression analysis was performed using normalized values of the 'bnchmk' variable and the computed factor scores F1 through F9. Additionally, a regression was performed using the 'bnchmk' variable and the set of independent variables which had been determined

using the S regression procedure, to be the best for constructing a model of response time. The results of both regressions are attached at appendices 2 and 3.

Extended Analysis Results

The regression involving 'bnchmk' and selected variables revealed that 'avm' (active virtual memory pages) alone in a model with a constant accounted for 53.9% of the variation in 'bnchmk' While all regression models constructed resulted in an overall F with significance at α < .005, all variables except 'avm' had an individual significance of .053 or greater. Examination of the computed variable correlation matrix revealed extreme multicollinearity problems involving all variables in the regression. Thus, the computed coefficients could not be considered very stable.

The regression using the computed factor scores as independent variables revealed that F1 (processing flow intensity), F2 (processing flow resistance), and F5(ul/temp filesystem utilization) in a model could account for 55.9% of the variance in 'bnchmk.' Again, all remaining models were significant at α < .005, however, significance of individual variables add after F1,F2 and F5 was .07 or greater. With the factor model, computed coefficients can be considered stable because all factors by definition are orthogonal and therefore, multicollinearity can not exist.

Thus, the removal of the possible inter-relationships between independent variables is an obvious advantage of performing regression with factors derived from a principal component analysis.

Part 2 - Classical Factor Analysis

Analysis Overview

The second phase of the overall factor analysis was to repeat the factoring process using classical factor analysis. This required specifying the PA2 parameter of the SPSS FACTOR procedure. The primary difference in the two techniques is the manner in which the unaccounted for part of the variance in each manifestation variable is handled. In principal component analysis this variance is ignored. However, classical factor analysis explicitly provides for an unexplained part of the variance in each manifestation variable as an error term, referred to as the specific variance. This assumption about the unexplained variance results in a different technique for computing communalities, which is the variance in each manifestation variable accounted for by the retained factors.

The primary reason for performing this analysis was to determine whether or not better interpretations would be possible for the computed factors. Because an iterative process is used to compute the communalites, it is possible that the resulting factors would be loaded differently by the manifestation variables, thus rendering a different interpretation of retained factors.

Analysis Results

As mentioned earlier, a reduced data set was used in classical factor analysis to allow computation of the deteriminant. As with principal component analysis, the initial factor matrix had the majority of the manifestation variables loading the first factor. Examination of the initial factor matrix showed that for the first three factors, the same variables loaded the same factors, with loadings for the classical factors, in most cases, being slightly less. Factors four and five were loaded oppositely by the two factoring techniques, with 'ulfs' and 'tmfs' loading classical factor four and principal component factor five. The remaining classical factors are not loaded above .5, thus interpretation of these would require considering significant loadings at a lower value. Even with lowering the significant loading value to .4, only factor six could be considered to have a significant loading. Thus, classical factors seven through nine cannot be easily interpreted due to insufficient loadings of manifestation variables.

An apparent reason for insignificance of factors seven through nine is the resultant communalities after iterations were completed. Convergence of communalities occurred after 12 iterations and the resulting factors and associated eigenvalues are shown in Table VI-4. Now, the first nine factors account for 100% of the variation in the

CPE data set, with only the first six having eigenvalues greater than one. The first six factors alone account for 93.3% of the variation in the manifestation variables. Thus, using the factor retention rule of thumb, only the first six factors express the dimensionality of the data set.

Table VI-4

Selected Classical Factors

FACTOR	EIGENVALUE	PCT	CUM PCT
1	19.329	56.3	56.3
2	4.719	13.7	70.0
3	3.380	9.8	79.9
4	1.837	5.4	85.2
5	1.690	4.9	90.2
6	1.064	3.1	93.3
7	.995	2.9	96.1
8	.725	2.1	98.3
9	.597	1.7	100.0

Factors and associated variables for the VARIMAX rotated factor matrix are shown in Table VI-5. With the exception of the computed values of the loadings and the insignificant loading of factor 6, loadings for the first seven factors in the rotated factor matrix render the same interpretations given by rotated principal component factor

matrix. Since rotated classical factors eight and nine show no loadings over .37, they do not contribute to interpretation.

Table VI-5

Classical Factor Groupings

FACTOR 1		FACTOR 2		FACTOR 3	FACTOR 4
FRE	TROFF	R	FR	H1	B
AT	NROFF	W	ROOTFS	CPUSYAC	SYCL
H0	F77	AVM	REALT	CPUUSAC	CSW
INT	VERS	RE		AVIOAC	
CPUSR	EDITS	PIN		KAC	
CPSYS	SWMONS	PO		ENFS	
CPIDL	UTILS	SR		USRFS	
EXPS	LSFS	NUPROCS			
PC	BNCHMK				

FACTOR 5	FACTOR 6	FACTOR 7	FACTOR 8	FACTOR 9
UlFS	????	DE?	????	????
TMFS				
PI				

As a result of the similar variable loadings for the classical factors, interpretation of the factors 1 through 5 and factor seven are the same as those in Table VI-3.

CPE Interpretation of Analysis Results

The results of the factor analysis showed that the multi-faceted system activity for AFIT's SSC could be represented by at least six to nine latent processes or factors. The software monitor and accounting data which was collected from the system represented three days of activity in terms of 52 variables. Principal component analysis reduced the CPE data set to nine factors, while classical factor analysis reduced it to six significant ($\lambda \geq 1$) factors. The factors produced tended to characterize the data set in terms of general types of processing activity and processing flow characteristics. In both factor analyses, the VARIMAX rotated factors one and two were loaded heaviest by the manifestation variables. The resulting interpretation of factor one as 'processing flow intensity' and factor two as 'processing flow resistance' implied that the system performance could be modeled and analyzed in terms of these factors.

To investigate the possible modeling use of the computed factors, the regression performed confirmed the hypothesis that factors one and two provided good indicators of system performance, with approximately 60% of the variation in response time being explained by the models developed. The models constructed showed that increases in either of these factors would result in an increase in system response time which is a common sign of

performance degradation. Thus, a study of the manifestation variables which make up each of these factors and their corresponding system function would be warranted in trying to determine causes for poor system performance.

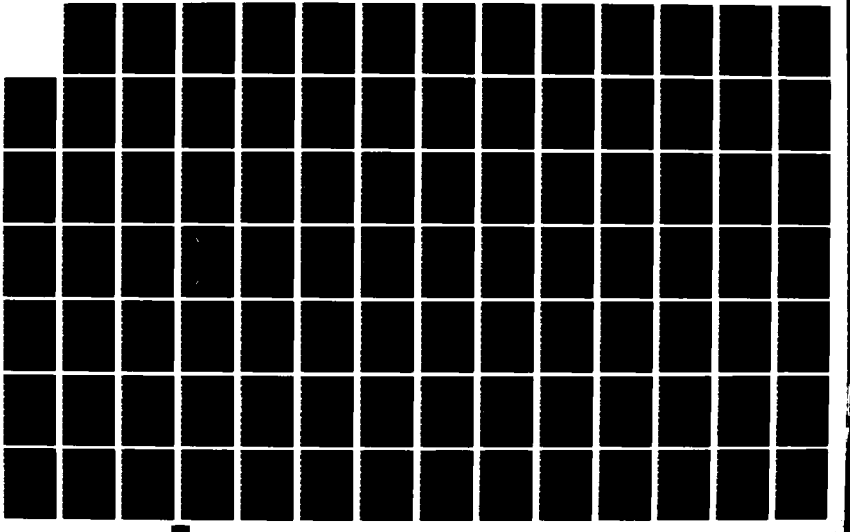
Conclusion

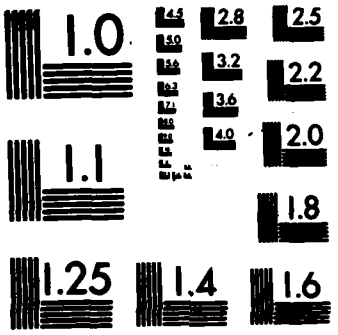
The ability to reduce the dimensionality of a data set with a large number of interdependencies is the primary utility in factor analysis. When the CPE analyst has a conglomerate data set of workload and performance parameters, chances are that redundant information is present. In the 52-variable data set used for the analysis done in this chapter, numerous parameters measured the same aspect of system activity. For example, cpusr, cpsys, cpidl, cpuusac, and cpusyac all measured cpu utilization. The first three were generated by a software monitor while the last two were taken from the system accounting file. Use of the two factor analysis techniques allowed 79 to 100 per cent of the variability in the data to be explained by nine independent factors. Once factors are computed, they can be named for interpretation by using one or more of the axis rotation techniques. Depending on complexity in the data (i.e. the number of variables that load more than one factor), named factors may or may not have specific meaning. Instead a more general description of the factor may be required.

After the factors have been interpreted, they can be

AD-A138 268 APPLICATIONS OF MULTIVARIATE STATISTICAL TECHNIQUES FOR COMPUTER PERFORMANCE EVALUATION(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 2/3

UNCLASSIFIED G L BRUNDIDGE DEC 83 AFIT/GCS/EE/83D-4 F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

used just like variables. Thus the computed factor scores for a given data set can be used in a regression. The primary advantage of using the factors is the absence of any interaction. By definition, factors are independent. Therefore, depending on how well the factoring techniques reduce the data set, and how well they allow interpretation, factor analysis can provide the analyst with a more compact means of examining numerous workload and performance parameters.

CHAPTER VII

DISCRIMINANT ANALYSIS

Analysis Objective

When it is suspected that there is an underlying structure to a set of data it may be advantageous to try and identify this structure via discrimination and classification techniques. Suppose observations in a data set consisting of performance/workload measures could be grouped by some aspect of system activity of job execution. It would be helpful to the analyst to be able to quantitatively identify these groupings using disriminating functions. Likewise, it would be useful to be able to examine how well new observations fit into the identified groups via use of derived classification functions. One means of accomplishing this is discriminant analysis.

The primary objective of this analysis is threefold. The first step in the analysis was to determine the nature of selected variable groupings in the CPE data set and to examine how well derived discriminant functions can distinguish between these groups. Second, derived classification functions were examined to determine how well they classified observations. Third, the significance of differences in group centroids was examined using

multivariate analysis of variance.

A dummy variable was established to create groups used in the analysis. The dummy variable used was 'resptim' and it was based on the variable 'bnchmk' which is an indicator of system response time. Three groups were created based on the following values of 'bnchmk':

Group 1 (resptim = 1) --> 0 - .2999
(good response time)

Group 2 (resptim = 2) -->.3 - .4999
(fair response time)

Group 3 (resptim = 3) -->.5 - 1
(poor response time)

Each of the groups represented a system performance condition with respect to response time. It was hoped that the discriminant functions derived by the analysis would provide insight into the importance that each of the selected discriminating variables played in separating the groups. This could in turn, reveal reasons for decreased system response time. Also, the derived classification functions could be used to determine the quality of response time that could be expected from the SSC, given a specific system performance scenario.

Analysis Results

The canonical discriminant function output for the response time group analysis is shown in Table VII-1.

Table VII-1

Canonical Discriminant Functions

FUNCTION	EIGEN VALUE	PCT OF VARIANCE	CUM PCT	CANON CORREL	AFTER FUNCTION	WILKS LAMBDA	CHI-SQUARED
					0	.030	211.099
1	10.964	85.77	85.77	.96	1	.355	62.188
2	1.819	14.23	100.00	.80			

As shown in the table, the chi-squared for function 1 is 62.188, which is significant at α = .001. The first function accounts for 85.77% of the variance in the discriminating variables and the second function accounts for the remaining 14.23%. The corresponding canonical correlations (.96 and .80) indicate that the discriminant functions are closely related to the 'resptim' group variable used to define the memberships of the three groups. Table 2 shows the values of the discriminant functions evaluated at the group centroids. The separations between the centroids indicate that the

Table VII-2

Discriminant Functions Evaluated at Centroids

Group	Func 1	Func 2
1	-1.82170	.02418
2	.43583	-2.00447
3	6.92589	4.90479

discriminant functions provide for good discrimination between group centroids. The strong centroid separation teamed with the function significance reveal that the discriminant functions provide a good means of distinguishing between the three response time groups.

Examination of the standardized discriminant function coefficients shown in Table 3, revealed that 'avm' (active virtual memory pages) made the greatest contribution to the first function with a coefficient of 7.7556. 'Po' (pages paged out) made the greatest contribution to the second function with a coefficient of -6.1343. Included in Table 3, are the unstandardized coefficients which are useful for further computational purposes, since they have not been adjusted for measurement scales and variability in the original variables.

Using the standardized coefficients shown in Table 3 and the varimax rotated coefficients provided in the SPSS output, names were given to the discriminant functions. The

first function could be called 'virtual memory use' because of the strong impact of 'avm.' The second function could be called 'page removal activity' because of the strong impact of 'po'. Thus, it can be concluded that function one discriminates between response time groups primarily based on virtual memory being used by currently running processes, while function two discriminates based on the number of pages being removed from primary memory.

After examination of the discriminant functions was completed, the derived classification functions were examined for validity. Table 4 shows the classification function coefficients derived using the 33 selected discriminating variables.

Table 5 contains the actual classification results. The results show that 89.96% of the cases were correctly classified when cases used in the analysis were classified. The percent correctly classified when cases not used in the analysis were used dropped to 53.13%. Thus, while the discriminant functions were quite useful for distinguishing between response time groups, the derived classification functions render just over a 50% correct classification rate. This means that the classification functions that were computed cannot be fully relied upon to correctly classify new observations.

Table VII-3

Discriminant Function Coefficients

Variable	Standardized Coeff.		Unstandardized Coeff	
	Func 1	Func 2	Func 1	Func 2
R	-1.562	1.014	.290	.144E-02
B	3.847	- .518	5.336	2.491
W	1.976	.087	145.635	102.795
AVM	7.56	-1.553	.571E-02	.223E-02
FRE	3.728	.677	.490E-02	.457E-02
AT	-1.819	-1.314	-1.182	-3.008
PIN	5.109	- .913	3.432	1.426
PO	-5.329	-6.134	- .899	-6.171
FR	-3.534	5.344	-4.124	1.821
INT	- .371	1.870	- .475E-01	.493E-01
SYCL	-2.322	.224	- .576E-02	- .296E-02
CPSYS	-3.247	- .704	- .240	- .239
NUPROCS	-3.591	- .097	- .108	- .733E-01
ROOTFS	1.201	.522	7.902	11.805
LSFS	- .753	- .322	- .844	-1.244
ENFS	2.451	- .085	2.129	1.265
U1FS	-1.850	- .047	-1.331	- .903
EXPS	4.052	2.832	.603E-02	.147E-01
REALT	1.310	1.277	.291E-01	.126
CPUSYAC	3.275	1.208	11.265	14.915
AVIOAC	-3.146	-1.332	- .649E-01	- .950E-01
KAC	- .075	.614	- .532E-01	.643E-02
TROFF	-1.124	- .647	- .224	- .431
KAREL	.585	.102	.134	.123
F77	-1.322	- .926	- .596E-01	- .145
VERS	- .882	.731	- .165	.201E-01
CC	.197	.758	- .136	.417E-01
PI	.522	.381	.328E-01	.845E-01
PC	- .219	-1.305	.335E-01	- .783E-01
SWMONS	- .420	- .124	- .188E-01	- .218E-01
SAPROC	.244	- .383	2.123	- .978
SRUN	.845	.142	.469	.426
UTILS	1.582	- .247	.252E-01	.111E-01
(CONSTANT)	0.0	0.0	-423.048	-555.939

Table VII-4

Classification Function Coefficients

CLASSIFICATION FUNCTION COEFFICIENTS
(FISHER'S LINEAR DISCRIMINANT FUNCTIONS)

RESPTIM =	1	2	3
R	267.2635	266.6053	264.7315
B	3463.530	3470.524	3522.365
W	-40531.18	-40410.94	-38755.52
AVM	4.821577	4.829932	4.882423
FRE	4.774033	4.775832	4.839227
AT	-2246.173	-2242.740	-2271.196
PIN	2632.309	2637.163	2669.286
PO	-5740.707	-5730.216	-5778.689
FR	282.5018	269.4973	255.3145
INT	3.094469	2.887365	2.919925
SYCL	-6.794753	-6.801755	-6.859615
CPSYS	-192.2012	-192.2574	-195.4753
NUPROCS	-208.2490	-208.3434	-209.5480
ROOTFS	34885.73	34879.62	35012.47
LSFS	-750.2603	-749.6410	-763.7170
ENFS	1903.404	1905.645	1928.204
U1FS	-876.8784	-878.0512	-892.9305
EXPS	4.574522	4.558387	4.698832
REALT	99.81454	99.62531	100.6819
CPUSYAC	13373.31	13368.48	13544.65
AVIOAC	-88.42795	-88.38185	-89.45993
KAC	-5.960621	-5.985689	-5.975805
TROFF	70.36670	70.73673	66.30674
KAREL	164.1805	164.2336	165.9542
F77	-82.51950	-82.35920	-83.75130
VERS	-31.74120	-32.15495	-33.08844
CC	359.6933	358.5386	360.5313
PI	68.42839	68.33102	69.12819
PC	5.553295	5.787792	5.463933
SWMONS	-13.89146	-13.88975	-14.16234
SAPROC	-2916.360	-2909.583	-2902.561
SRUN	935.1774	935.3733	941.3640
UTILS	33.41976	33.45402	33.69399
(CONSTANT)	-872979.9	-872808.1	-879429.7

Table VII-5

Classification Results

THESIS DISCRIMINANT ANALYSIS - RESPONSE TIME
CLASSIFICATION RESULTS FOR CASES SELECTED FOR USE
IN THE ANALYSIS

ACTUAL GROUP	NO. OF CASES	PREDICTED GROUP MEMBERSHIP 1	2	3
GROUP 1	44	42	2	0
		95.5	4.5	0
GROUP 2	25	6	19	0
		24.0	76.0	0
GROUP 3	10	0	0	10
		0	0	100.0

PERCENT OF GROUPED CASES CORRECTLY CLASSIFIED - 89.87

CLASSIFICATION RESULTS FOR CASES NOT SELECTED FOR USE IN
THE ANALYSIS

ACTUAL GROUP	NO. OF CASES	PREDICTED GROUP MEMBERSHIP 1	2	3
GROUP 1	22	17	2	3
		77.3	9.1	13.6
GROUP 2	35	15	15	5
		42.9	42.9	14.3
GROUP 3	7	1	4	2
		14.3	57.1	28.6

PERCENT OF GROUPED CASES CORRECTLY CLASSIFIED - 53.13

CLASSIFICATION PROCESSING SUMMARY

To conclude the response time group analysis, selected discriminating variables were used in a one-way multivariate analysis of variance (manova) with the response time groups defining the treatment levels. With manova, the objective was to determine whether there was a significant difference among group centroids. Both multivariate and univariate tests of significance indicated that very strong differences exist between group centroids. Under multivariate tests, Hotellings T^2 equals 734028.1431, with an approximate F of 3246662.9355. Since the F is significant at α = 0, for Hotellings T^2 and the remaining two statistics, the null hypothesis that the centroids are equal can be strongly rejected. Thus, it can be concluded that groups defined by 'resptim' have significantly different centroids.

CPE Interpretation of Results

The discriminant analysis showed that the variables which represented system performance and workload indicators for the SSC could be separated into at least three distinct groups based on the system response time indicator 'bnchmk.' Based on the standardized coefficients for the derived discriminant functions, it can be concluded that the primary discriminating variables were 'avm' (for function one) and 'po' (for function two). The fact that 'avm' and 'po' are both virtual memory related variables

indicates that the level of memory use on the SSC will have a significant impact on system response time. When current processing requires extensive use of virtual memory, response time will most likely be degraded. Thus, further investigation of memory allocation and use is warranted to gain insight into exactly why high virtual memory activity causes degraded response time.

The manova served to further confirm the fact that different system performance scenarios, as reflected by different values of the selected discriminating variables, result in distinct groups with significantly different group centroids. These group centroids could be investigated to determine what values of the discriminating variables, on the average, result in a given response time group. Variables that show extreme changes can then be further examined for additional insight into why response time changes.

Conclusion

The use of discriminant analysis has its primary value in being able to determine the significance of group differences. As an additional feature, classification functions can be derived that will allow the analyst to test how well observed system activity can be classified into the groups of interest. Unlike nominal or ordinal level data, where groups can be defined for single values

of a selected variable, most CPE data sets require that groups be defined based on specified intervals of a selected performance or workload parameter. The parameter chosen to distinguish groups should be a parameter that can be theoretically related to the manifestation variables. This theoretical relationship can then be used to determine what effect a manifestation variable has on the discriminating variable. Subsequently, it may be possible to determine how significant the effect of the manifestation variable is, based on how well it discriminates between groups. Thus, as an alternative to regression, discriminant analysis can be used to analyze the effect a parameter or group of parameters has on another parameter.

CHAPTER VIII

CLUSTER ANALYSIS

In data analysis the study of relationships within a given data set need not always be based on the existence of strictly quantifiable relationships. Situations may exist where the analyst desires to know what type of interrelationships exist among variables or observations in a data set based only on the values that these variables or observations take on. Thus, no requirements for dependence or independence exist and few or no assumptions are necessary to perform an analysis of data in this manner. One possible goal of an analysis of this type is to see how variables/observations will group or cluster based on the values they take on. Subsequently, the analyst can examine the clusters with questions as to why clusters formed in a specific order or with a certain membership. A group of analysis techniques collectively called cluster analysis allows the analyst to perform this type of less restrictive examination of his data.

As stated in the discussion on cluster analysis, objectives for using various clustering techniques on data can be quite varied depending on characteristics of the data, purposes for studying the data and analyst insights into the data. The 143 performance summary intervals (PSIs)

which make up CPE data set one, are each characterized by 52 variables. Thus, available clustering techniques allow for a number of different analyses to be performed. For purposes of this research, K-means clustering was used to perform a non-hierarchical cluster analysis of selected subsets from CPE data set one.

The selected data subsets consisted of a combination of one or more of the software monitor measures and all of the monitored command frequency counts for each PMI. The software monitor measures used included;

1) Memory Management Measures - re, pin, po, fr, sr

2) Cpu Utilization Measures - cpuusr, cpsys

In addition to the measures shown, one or more of the following variables was included in each of the cluster analyses:

1) BNCHMK - indicator of system response time

2) NUPROCS- count of currently running user processes

3) EXPS - number of processes executed during a PMI

It was hoped that clusters formed would provide insight into the relationship between various system activity levels, as given by the software monitor measures, and the types of processes which were being executed during the associated PSI's. Those insights could in turn provide a basis for constructing performance hypothesis related to the impact of different types of processes on specific aspects of system activity.

The K-means clustering program used in this analysis is a BMDP (PKM) subprogram and provided statistical and graphical descriptions of each developed cluster. After the clusters were generated for each of the selected data subsets, attention was first devoted to the report of cluster variable means for cases contained in specific clusters. Secondly, ANOVA measures were reviewed to determine the strength of differences in variable values contained in each cluster.

Analysis Results and Interpretation

The first software monitor measures analyzed were those which made up the cpu utilization measures. Four clusters were requested in an attempt to determine whether or not four distinguishable clusters could be formed. Shown below are the cpu utilization means for the 4 clusters formed.

Cluster 1: cpusr = 42.36 cpsys = 24.37

Cluster 2: cpusr = 49.17 cpsys = 20.73

Cluster 3: cpusr = 18.71 cpsys = 6.99

Cluster 4: cpusr = 64.09 cpsys = 31.24

The cluster means reveal that there are two strongly separated clusters (3,4) and two clusters which consist of similar cases (1,2). Cluster 3 clearly consists of low cpu utilization PMIs with the mean overall utilization (cpusr + cpsys) being 25.70 per cent. The bnchmk and nuprocs

variables were also at their lowest in the PMI's contained in cluster 1. Examination of the cases contained in the cluster revealed that they were primarily PSIs from the early morning hours when system activity is characteristically at a low level.

Cluster 4 is at the other extreme with an overall cpu utilization of 95.33 per cent. The cases which are closest to the center (those which most strongly characterize the cluster) are PMIs from prime time processing hours (0800 - 1800) and evening hours (1800 - 2300) on day2 and day3. The absence of PMIs from day1 is probably because that day was a Sunday when system activity is relatively low. From examining just these clusters, it was possible to begin characterizing time periods when the system was most utilized. Means for bnchmk and nuprocs were at their highest, with bnchmk being 50% greater than the overall average of .318 seconds and nuprocs being 147% greater than its overall average of 30.8. These increased values of bnchmk indicate that during PMIs contained in cluster 4 the system began experiencing performance degradation in the form of increasing response time.

Examination of the types and counts of selected processes being run during the PMIs of cluster 4 show that counts for f77, vers, edits, cc, srun and utils were all up at least 60% over their overall mean values. If the overall means for the variables in the CPE data set used could be

variables were also at their lowest in the PMI's contained in cluster 1. Examination of the cases contained in the cluster revealed that they were primarily PSIs from the early morning hours when system activity is characteristically at a low level.

Cluster 4 is at the other extreme with an overall cpu utilization of 95.33 per cent. The cases which are closest to the center (those which most strongly characterize the cluster) are PMIs from prime time processing hours (0800 - 1800) and evening hours (1800 - 2300) on day2 and day3. The absence of PMIs from day1 is probably because that day was a Sunday when system activity is relatively low. From examining just these clusters, it was possible to begin characterizing time periods when the system was most utilized. Means for bnchmk and nuprocs were at their highest, with bnchmk being 50% greater than the overall average of .318 seconds and nuprocs being 147% greater than its overall average of 30.8. These increased values of bnchmk indicate that during PMIs contained in cluster 4 the system began experiencing performance degradation in the form of increasing response time.

Examination of the types and counts of selected processes being run during the PMIs of cluster 4 show that counts for f77, vers, edits, cc, srun and utils were all up at least 60% over their overall mean values. If the overall means for the variables in the CPE data set used could be

considered as representative of good system performance then based on these clusterings one could hypothesize that increases in process counts of 60% over these averages results in a 50% increase in response time. In an actual performance evaluation further data could then be collected and analyzed, using other CPE tools, to test this hypothesis.

The next set of software monitor measures to be examined were those related to virtual memory management. The mean values of the five measures involved for the four clusters formed are shown in Table VIII-1.

Table VIII-1

Cluster Means for Memory Measures

	re	pin	po	fr	sr
Cluster 1	7.32	2.79	2.06	2.29	15.00
Cluster 2	2.19	1.13	.74	.58	4.70
Cluster 3	.06	.11	.01	.01	.13
Cluster 4	10.67	2.82	2.56	2.18	21.14

Unlike the clusters for cpu utilization, these clusters have a definite level structure, with 3, 2, 1, 4 being the cluster ordering for increasing levels of activity. Cluster 3 almost exclusively contains PMIs which are from day1 (Sunday) and the early morning hours of day2 and day3 (0100-0700). Cluster 2 PSIs cannot be characterized by any specific time period. However, the PSIs which are contained

in this cluster are those which have system activity levels that are closest to the overall system mean level. Moving to cluster 3, significant increases began to take place. Paging activity (pin, po), page reclaims and frees (re, fr) and page scanning by the implemented clock algorithm all increased by at least 40 to 50 percent over the overall mean. The response time indicator, bnchmk, increased by 19%. While all process counts for monitored processes increased, sharp increases to peak values took place with karel, srun and othprocs (the difference between all executed processes,exp, and the sum of all monitored processes).

Cluster 4 represents the highest level and with the exception of pi, saproc and those processes which peaked in cluster 3, all other monitored processes peaked in this cluster. Bnchmk had an average value of .4791, which is a 51% increase over the overall mean. Thus, with the high activity in memory management activity came degraded performance (as compared to the mean) and a consequent increase in response time. As with the clusters for cpu utilization, these clusters indicated that the level processing at which system performance begins to degrade occurs when the number of processes increases by at least 50% over the usual system average (assuming the system average used is defined as being indicative of acceptable performance). In an actual performance study the

hieracrchical clustering technique could be used to cluster values of individual variables (performance activity monitors or process counts). The results of the variable clustering combined with what was revealed in the case clusterings such as the two above could provide insight into the relationship that exists between a particular variable and particular level of performance.

Conclusion

The utility of cluster analysis, is in it versatility and robustness. The class of algorithms which make up the available cluster analysis techniques provide numerous approaches to studying inter-relationships within a data set. While the analysis performed here extracted performance and workload information based on cluster composition and statistics, the analysis of clusters can be extended. Possible extensions include, stepwise clustering, where observations or variables in initial clusters are again clustered with the same or additional objectives for cluster examination. Observations in the cluster can also be further analyzed using one or more of the analysis techniques examined in this study.

With respect to specific CPE applications, clustering is useful in examining the effects of system modification. Clusters generated before and after the modification can be examined using the approaches mentioned, with the hope that

any observed cluster differences can be related the modification. Therefore, as seen with the application of cluster analysis in this chapter, it is a data analysis technique which allows relatively unrestricted exploration of relationships that might exist in the data. Likewise, it allows the analyst to freely speculate about his observations as long as he realizes that in most cases, the clusters alone will not provide the means to fully test his speculations.

CHAPTER IX

CONCLUSIONS AND RECOMMENDATIONS

The use of multivariate analysis techniques to study and evaluate the peformance of AFIT's VAX 11/780 SSC proved to be quite successful. While the information rendered by each of the techniques was not in the form of a performance or workload model, it did provide insight into relationships that existed between performance and workload parameters measured in the data. The insight gained into the existence of relationships, their strengths, bases, and possible causes, could subsequently lead to the construction of testable hypotheses about different aspects of system performance and system workload. With the exception of OLS regression and ridge regression (which are not true multivariate techniques) the primary value of the techniques studied lies in the exploratory capability they provide to the CPE analyst.

It is a common occurrence in the CPE environment for an analyst to have available to him an abundance of system performance and workload data and not know exactly how to use it. The data is usually in the form of system accounting and software monitor data from files which are built by the system's accounting and monitor routines. In

many cases this accumulated data is either recorded and then stored in the tape library or left untouched by system analysts. However, results of this study show that the use of multivariate analysis on the data can provide a means of inexpensively monitoring certain measurable relationships of system performance and workload.

Data collected from AFIT's SSC consisted of accounting and software monitor data that was summarized and then combined into data sets that reflected a three and a four day period of system activity. Since fifty-two items of performance and workload measures were collected, the data set provided a good multivariate object of study. Primary results are summarized below for each of the techniques along with possible applications.

OLS Regression and Ridge Regression

Multiple linear regression is not a true multivariate technique. Even though a regression analysis involves more than one variable, the goal of the the analysis is to examine dependency of a single variable on the values of a number of other variables. OLS regression was performed in this study in an attempt to show its traditional use as a modelling technique in CPE and consequently highlight some of the problems that arise when using regression.

Results of the regression analysis rendered models that all presented an intuitive feel for how various system

activities were related to each other. For example, the 'runq' model indicated that the size of the process run queue is dependent upon the number of swapped processes, amount of memory being used, paging and disk activity, and the number of device interrupts being processed by the cpu. Thus, based on the signs of the coefficients one would expect the process run queue to increase in size as memory usage and disk activity increased and as swapped processes, paging activity, and device interrupts decreased.

However, assumption violation problems were revealed in the residual analysis, and as result, all of the models would have to be further developed before they could be used for reliable interpretation. For example, in the models for 'bnchmk'' and 'runq,' adding dummy variables to represent time periods of data collection could provide a possible solution to the apparent autocorrelation problem. Likewise, use of an X transform could reduce the heteroscedascticity problems which were apparent in the 'cpusr,' 'realt,' and bnchmk' models.

To remedy multicollinearity, ridge regression presented a means of regression modelling which used incremental biasing to negate the effect of strong correlations between independent variables on the stability of the regression coefficients. Models constructed using various amounts of bias showed an improved interpretive capability in the models. Thus, as an initial improvement

to a regression model, the CPE analyst could resort to ridge regression and examine whether a small amount of bias could be used to improve the explanative capability of a regression model. The analyst must remember, however, that the greater the amount of bias required the less the model will reflect the actual information contained in the data. As a result, models which contain highly correlated independent variables ($r > .80$) may require so much bias that the resulting model may still be of little interpretive use and completely useless as a predictive tool.

Canonical Correlation

In the situation where the analyst is not sure of how various system functions are related, a possible first step in analyzing performance/workload data would be to examine how selected groups of parameters are related to each other. For example, a possible interest would be how a group of measures for cpu utilization are related to measures of interrupt activity. For a study of this nature, canonical correlation would be useful. The groups of measures would be named and referred to as super variables and the canonical correlation analysis would show the correlation between the super variables. Thus, the analyst is able to see how one general aspect of system activity relates to another.

In the data sets constructed for the SSC, one of the super variables used was 'cpu utilization status,' which was a combination of five measures in the data set. This super variable was related to six other super variables constructed from the other related measures in the data set. Examination of the canonical correlations for 'cpu utilization status' revealed that the strongest super variable relationship existed with 'processor interrupt activity'. Variables included in this set include 'int' (device interrupts per second), 'sycl' (system calls per second) and 'csw' (context switch rate). The strong correlation between these two super variables (also called canonical variates) indicated that processing conditions which contain extensive processor interrupt activity resulted in high cpu utilization rates. Other strong relationships that existed revealed that increases in memory and disk activity (which are themselves related) also resulted in increased cpu utilization.

Factor Analysis

In many instances, collected performance and workload data contain redundant information. For example, a number of different variables in the data set may provide essentially the same information about cpu utilization or memory activity, or a subset of the data set variables together may reflect a latent underlying aspect of system

peformance or workload. To determine the true dimensionality of a CPE data set the analyst could use factor analysis to determine how much interdependency existed in collected data.

The results of the factor analysis in this study showed that the multi-faceted system activity for AFIT's SSC could be represented by at least six to nine latent processes or factors. The software monitor and accounting data which were collected from the system represented three days of activity in terms of 52 variables. Principal component analysis reduced the CPE data set to nine factors, while classical factor analysis reduced it to six factors. The two most significant factors produced tended to characterize the data set in terms of general types of processing activity and processing flow characteristics. The resulting interpretation of factor one as 'processing flow intensity' and factor two as 'processing flow resistance' implied that the system performance could be modeled and analyzed in terms of these factors.

To investigate the possible modeling use of the computed factors, an OLS regression was performed using computed factor scores for each factor as independent variables. The regression performed confirmed the hypothesis that factors one and two provided fair indicators of system performance, with approximately 60% of the variation in response time being explained by the

models developed. The models constructed showed that increases in either of these factors would result in an increase in system response time which is a common sign of performance degradation. Thus, a study of the manifestation variables which make up each of these factors and their corresponding system function would be warranted in trying to determine causes for poor system performance.

Discriminant Analysis

If the collected performance/workload data is structured so that observations can be grouped based on defined criteria, then information about a system may be gained by studying how distinct are the predefined groups. The analyst wishing to examine data in this manner can use discriminant analysis to obtain this type of information. In addition, group separation can be further analyzed using multivariate analysis of variance (manova).

The discriminant analysis showed that the variables which represented system performance and workload indicators for the SSC could be separated into at least three distinct groups based on the system response time indicator 'bnchmk.' Based on the standardized coefficients for the derived discriminant functions, it can be concluded that the primary discriminating variables were 'avm' (for function one) and 'po' (for function two). The fact that 'avm' and 'po' are both virtual memory related variables

indicates that the level of memory use on the SSC will have a significant impact on system response time. When current processing requires extensive use of virtual memory, response time will most likely be degraded. Thus, further investigation of memory allocation and memory use is warranted to gain insight into how high virtual memory activity causes degraded response time.

The manova served to further confirm the fact that different system performance scenarios, as reflected by different values of the selected discriminating variables, resulted in distinct groups with significantly different group centroids. These group centroids could be investigated to determine what values of the discriminating variables, on the average, result in a given response time group. Variables that show extreme changes can then be further examined for additional insight into why response time changes.

Cluster Analysis

With the exception of factor analysis, the application of all the previous techniques required existence or imposition of some underlying structure in the data. It may be desirable in an analysis to simply examine standardized or non-standardized distance/similarity relationships that exist within collected CPE data. In this situation the analyst can use one or more of the available clustering

techniques to study these type of relationships between variables or observations.

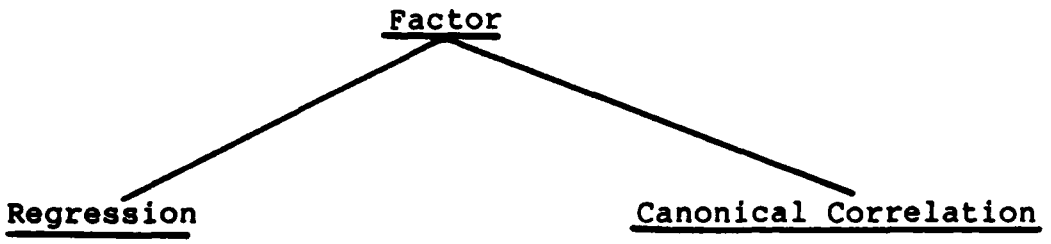
The application of cluster analysis showed that it is a data analysis technique which allows relatively unrestricted exploration of relationships that might exist in the data. Likewise, it allows the analyst to freely speculate about his observations as long as he realizes that in most cases, the clusters alone will not provide the means to fully test his speculations. Examination of the data collected from the SSC revealed that clusters could be formed to examine how a particular type/amount of processing affected various measures contained in the data. The two sets of measures examined included cpu utilization measures and virtual memory activity measures. In each case four clusters were formed. For cpu utilization, two of the clusters reflected the extremes in processing activity, while the memory activity measures clustered in four distinct levels of system activity. Examination of the counts for various types of processes which were run during the PMIs contained in each cluster enabled conclusions to be drawn about the impact of different types of processes on overall system activity level.

Consolidated Analysis Approaches

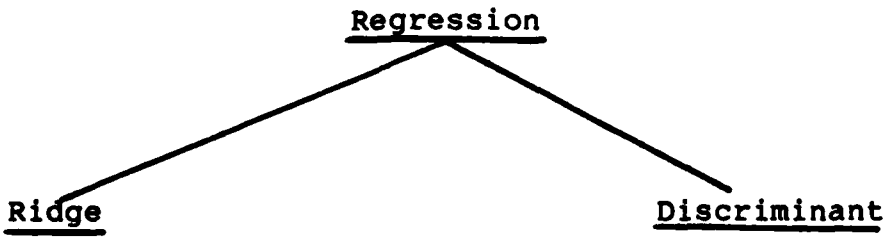
One of the primary objectives of this thesis effort was to evaluate each of the multivariate analysis techniques for CPE applications. As a result, the presentation thus far has concentrated on the techniques as individual tools. However, if the analyst has a number of these techniques available to him, then a consolidated approach to data analysis may be taken. Each of the multivariate techniques differ in the type of information that is extracted from the data. Using one technique to further investigate or improve results of another could lead to more accurate performance evaluation information.

The selection of which techniques to use in a CPE study will depend on availability and the objectives of the analysis. In this study, consolidated approaches were used in the regression and factor analysis chapters. OLS regression was followed up with a ridge regression to build a more explanative model. The interpreted factors derived from the factor analysis were regressed to determine how well they could explain variation in a data set performance parameter. While only two techniques were used in these approaches, other consolidated approaches could involve as many as all of the techniques. Figure IX-1 shows three possible approaches involving two or more of the techniques.

APPROACH 1



APPROACH 2



APPROACH 3

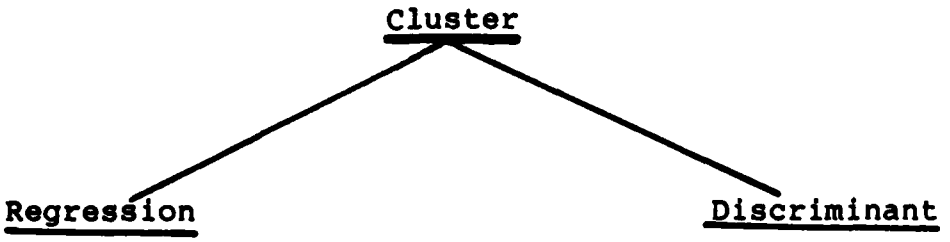


Figure IX-1. Possible Consolidated Approaches

In APPROACH 1, the analyst begins by trying to reduce the dimensionality of his collected performance data. If he is successful in deriving a set of interpretable factors, he can then regress the factors against performance or workload parameters in the data set. The regression could lead to a model of the parameter that is free of multicollinearity. This would provide a good explanation of the dependency relationships between the criterion and predictor variables. If the factors are not interpretable, then variables which make up the factors can be grouped into super variables and a canonical correlation can be done using selected pairs of supervariables. Calculating redundancy measures for the two super variables (canonical variates) could aid in interpreting the groupings.

When the analyst's goal is modelling parameters in a well understood data set, APPROACH 2 could be used. OLS regression is used first to model a selected workload or performance parameter. Based on how well the model meets the regression assumptions, the analyst can choose to use the OLS model. If the OLS model violates the multicollinearity assumption then ridge regression could be used to improve coefficient stability and provide a better explanatory model. If further examination of the dependent variable is desired, the analyst can define groupings of the independent variables based on designated intervals of the dependent variable and then perform a discriminant

analysis. The independent variables are now labeled manifestation variables and will be examined to see which of them has the strongest influence in determining group membership.

Finally, an analyst using APPROACH 3 is again faced with a large data set with which he has very little familiarity. However, in this case, the goal is not to reduce dimensionality (the goal in APPROACH 1). Instead, he desires to examine how the multivariate observations group based on a selected similarity measure such as distance from a specified or derived centroid. Applying a non-hierarchical clustering technique to cluster cases (i.e. the K-means cluster program used in this study), the analyst could then examine cluster membership. Ideally, he would have specific parameters selected for comparison between clusters so that variations in cluster membership would have some significance. The cases which make up clusters are essentially a new data set with characteristics that cause them to be separable from the other cases in the full data set. Subsequently, cases in each of the clusters could be further analyzed using any of the techniques presented in this study. In the example approach given in Figure IX-1, either regression or discriminant analysis could be used to examine selected dependency relationships within specific clusters.

It is easy to see that the number of consolidated

approaches available to the CPE analyst are many in number. Also, the approaches can vary in component techniques as much as the performance and workload environment in which the analyst functions. Thus, while not one consolidated approach is applicable to all CPE situations, given the performance data, at least one approach should exist that would allow the CPE analyst to explore the data for possible peformance/workload information. In the section that follows information gained from two or more of the techniques studied was used to formulate two of three performance hypotheses presented. This should provide a better appreciation for using consolidated approaches with multivariate techniques.

SSC CPE Observations

While sole use of the multivariate analysis techniques did not allow a complete performance evaluation to be performed on the SSC, information gained from applying the techniques did provide the basis forming two of the performance hypotheses presented on the following pages.

Hypothesis #1

** The SSC is a virtual memory machine that peaks in performance when processing requirements raise virtual memory use to the point where the size of the freelist decreases to a system defined critical threshold. At this point the overhead involved in memory management (i.e. paging, swapping, page scanning etc.) becomes a significant part of the system workload. In intense processing situations, where the number of user processes is high, resulting in high memory demand, the memory management overhead becomes so significant that contention with user processes results. Since the UNIX system processes have priority when contending for resources, the execution rate of user processes is retarded and subsequently system performance is degraded. The user sees the degraded performance in terms of increasing response time.

Hypothesis justification:

--Ridge regression model indicated that the number of user processes along with size of the process run queue and the number of editor process (vi,ed,emacs,sed) were resulted in or coincided with the largest increases in active virtual memory use.

--The discriminant analysis identified active virtual memory as one of the key discriminators between response time groups.

--The cluster analysis showed high virtual memory use occurred when the number of user processes was high. It also showed that PSIs with the highest virtual memory use had slowest response time.

--A program (vmgraph) developed to represent vmstat data in graphical form on a near real time basis showed memory management activity (pi,po,re,fr,sr) sharply increased whenever size of the free list fell below 900 pages (page = 1024bytes).

Hypothesis #2

** Degraded performance can not be attributed to a single type of process. When workload reaches the point where noticeable degradations in performance occur, the workload is usually characterized by a uniform increase in the number of all processes being executed. Thus, performance degradation on the SSC is more a problem of process quantity than process quality.

Hypothesis justification:

--None of the analyses performed identified any one process as being constant cause of hampered performance.

--Cluster analysis showed that PSIs where all processing resources were being heavily used, had a process mix that was on the average 40 to 90% greater than usual accross all types of processes.

--Factor Analysis showed that the number of user processes was a key element of the 'process flow resistence' factor, while all of the monitored process types contributed to the 'process flow intensity factor.

--Neglecting consideration of the violated assumptions, none of the OLS regressions which modeled key workload or performance paramenters (i.e. runq,avm,cpuusr,bnchmk,cpsys) identified any one specific process type as having a significant impact across all models.

--Canonical Correlation revealed that no process more significantly loaded the process mix super variable (canonical variate). All frequently executed processes loaded the variate with relatively equal correlations.

Hypothesis #3

** Because UNIX allows background processes the the number of users currently logged in will generally not be indicative of user generated workload. This capability teamed with the fact that the system is managed primarily as a free resource to all users, results in frequent user abuse of processing and storage resources.

Hypotheis justification:
(primarily based on observation)

--Observation revealed that the number of user processes would exceed the number of logged in users by 50 to 100 percent.

--Descriptive statistics showed that the number of user processes was never 0, regardless of the time of day.

--During course of this study there were frequent occurrences of file system overflows and multiple submission of identical background processes.

Conclusion

The use of multivariate analysis can provide the CPE analyst with a useful alternative/supplement to regression modeling. The ability to explore the structure, dimensionality and relationships that exist within collected performance and workload data could provide the analyst with a means to form hypotheses about various aspects of system performance. In some situations the hypotheses would be testable based on the results given from the multivariate technique, while other situations would require further examination of the collected data or collection of additional data. Thus, in addition to the traditional use of regression analysis to study collected performance and workload data, use of multivariate analysis techniques give the CPE analyst an additional dimension in his ability to analyze performance related information that may be contained in collected CPE data.

Recommendations

In this study, six analysis techniques were used to examine collected performance/workload data for system performance information. The target system was the AFIT VAX 11/780 SSC, which provided an abundance of peformance and workload data sources. While the approach used here primarily surveyed application and usefulness of each of the techniques, further study is warranted in using one or more of the techniques in conjunction with other CPE tools

to perform a more complete performance evaluation of SSC or other selected target systems. In addition to using system generated data, the nature of multivariate analysis allows inclusion of other non-system related variables which impact system performance. Thus, a workload study of the SSC could also include data which reflects current course offerings which require SSC support, the type of support required, and quarter mid-term periods and major project due dates. Using the possible problem areas revealed in this study as a starting point, further study of the VAX hardware and UNIX operating system teamed with the use of traditional CPE tools (i.e. analytical or simulation modelling) and multivariate analyses, could enable a thorough performance evaluation of the SSC to be performed. This would allow the potential for performance improving hypotheses to be formulated, tested and possibly implemented.

Learning Experiences

Learning experiences from this thesis effort were many. Fundamental understanding of univariate and multivariate statistics was greatly increased because of the application oriented problem studied in this effort. Working many hours on the SSC afforded time to become comfortably familiar with UNIX operating system and many of its conveniences. Also, having been required to use the ASD

CYBER to perform most of the analyses enabled knowledge to be gained about more than one computer system and how to efficiently use two very different operating systems. While all of the forementioned were learning experiences, the most useful knowledge gained from this effort was the role that a CPE analyst must play in evaluating a systems performance.

Unlike the software designer/analyst, or the computer hardware specialist the performance analyst does not specialize in any one area of a computer system. Instead, he must approach his job as a generalist, with at least a global familiarization with all major components of the system to be analyzed, hardware and software. In addition to the system knowledge required, skills at developing software utilities and using different types of hardware are almost essential. During this study, playing the role of the performance analyst was both challenging and rewarding and provided the most gratifying and useful learning experience.

APPENDIX A

THE VAX 11/780 SYSTEM ARCHITECTURE

AND

UNIX OPERATING SYSTEM

Part 1 - The Vax 11/780 Architecture

AFIT's Scientific Support Processor is a VAX 11/780 super-minicomputer. Introduced by Digital Equipment Corporation in 1978, the VAX was designed in part to remedy the limited addressing capability of its predecessors in the PDP-11 family. (Ref 3:429-432) However, the overall goal for design was to provide a product with more overall performance and functionality than the PDP-11, while maintaining PDP-11 compatibility. The VAX was developed as a system, with all hardware and software development groups working together to ensure efficiency of the integrated system.

Global System Structure

Figure A-1 (Ref 19:25) is a block diagram which shows the VAX's global system structure. As shown in the figure,

a primary component of the system is the Synchronous Backplane Interconnect (SBI) which is an internal synchronous data and control bus that operates on a 200 ns period. Connection to mass storage devices and other peripherals is provided via the MASSBUS and UNIBUS bus systems. The VAX 11/780 can support up to four UNIBUS and four MASSBUS adapters. Figure A-1 also shows the relative simplicity of the VAX structure. Thus, the improved capability of the VAX did not come at the cost of simplicity in architectural design. In addition to the SBI, other major system components include, 1) the processor, which contains the CPU, a cache and a writable diagnostic control store, 2) primary memory, 3) MASSBUS and UNIBUS adapter(s) and 4) the Q-bus.

Figure A-2 (Ref 3:460) shows the global system structure of the VAX using a PMS description. The following description information corresponds to the parenthesized numbers.

---

(1) Pc('VAX 11/780; microprogrammed; 8-byte instruction buffer; data-path: 32 bits; 1 byte: 8 bits; 1 word: 2 bytes; 1 longword: 4 bytes; 1 quadword: 4 words; variable length instructions: 1:n bytes; 16 long word general registers; Mps = apprx 17 longwords; native and PDP-11 emulation instruction sets)

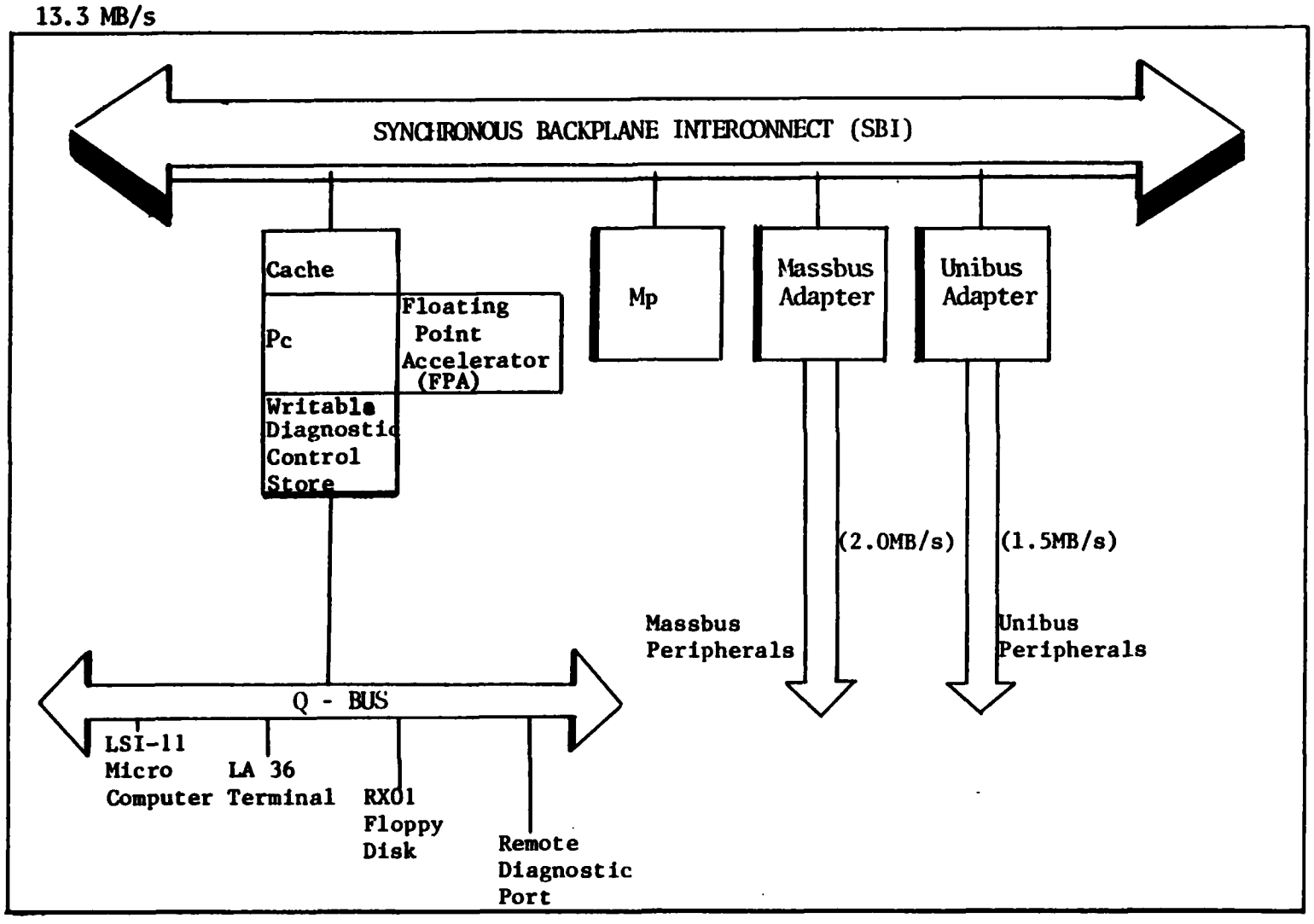


FIGURE A-1 VAX ARCHITECTURE (Block Diagram)

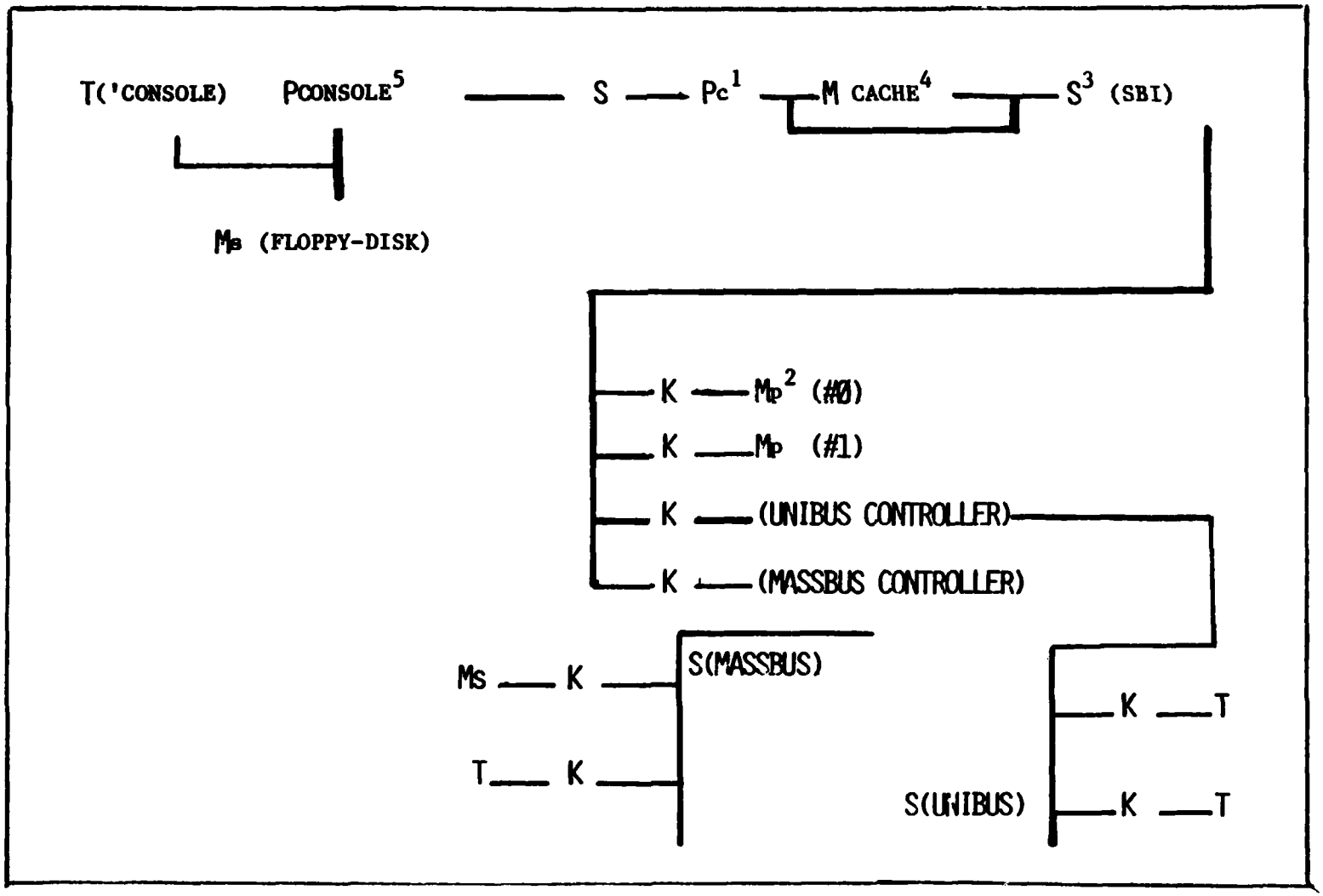


FIGURE A-2. VAX ARCHITECTURE (PMS)

(2) Mp(ECC MOS; 1M bytes -- 8M bytes; i-unit: quadwords; 8 bits ECC; t.cycle:600ns; t.access:1800ns; longword addressable)

(3) S('Synchronous backplane interconnect; 30 address lines (half Mp, half I/O); 32 data lines; t.cycle: 200ns;transfer rate: 13.3M bytes/second)

(4) Mcache(8K bytes; t.cycle: 200ns; set associative)

(5) Pconsole('LSI-11)

The Central Processor Unit

The VAX 11/780 processor has a 32 bit architecture based on the PDP-11 family of 16-bit minicomputers. The CPU is microprogrammed, and has a writable control store which can be loaded from the console subsystem for diagnostic purposes (Ref 3:460). It also contains an 8-byte instruction look ahead buffer to increase processing speed. Two separate D-units are used to perform shifting operations, and integer and floating point arithmetic in parallel. To achieve the required PDP-11 compatibility, the microprogram can interpret two instruction sets, the VAX native mode instructions and the PDP-11 compatibility mode instructions.(Ref 3:461; Ref 6:24)

The CPU uses multiple address modes and stack structures similar to those of the PDP-11. However, the 32-bit addressing provides a large program address space plus 32-bit arithmetic and data paths for increased processing speed and accuracy. The processor uses 16 general registers for temporary storage, as accumulators, as index registers and as base registers. The registers are also used in the addressing modes to identify instruction operand locations. (Ref 21:407). The ISP description of the 11/780 CPU is shown in Figure A-3 (Ref 3:462).

The addressing modes referenced earlier are listed and defined in Figure A-4 (Ref 21:408). The modes are the same as the PDP-11 modes except no autodecrement deferred is implemented, and the literal, displacement and displacement-deferred modes are unique to the VAX (Ref 21:408). All of the modes except the register mode can be modified by an index register. When the index register is used, the mode is referenced as the basic mode with the suffix "Indexed." Thus, an additional six indexed addressing modes are recognizable by the processor.

R[0:15] <31:0>	16 general registers (although 4 have reserved functions as shown below and R[0] to R[5] can also be reserved for specific tasks)
PC := R[15]	Program counter
SP := R[14]	Stack pointer; there are 4 SP registers one for each kernel, executive, supervisor and user access modes
FP := R[13]	Frame pointer used for procedure calls
AP := R[12]	Argument pointer used in procedure calls
PSL <31:0>	Processor status longword
PSW <15:0> := PSL<15:0>	Processor status word
CC <3:0> := PSW<3:0>	Condition codes
T := PSW<4>	Trace bit
Trap bits := PSW <7:5>	Integer, floating and decimal overflows
Unused: PSW <15.8>	
IPL := PSL <20:16>	Interrupt priority level
Prev. mode :=PSL <23:22>	
Curr. mode :=PSL <25:24>	Kernel, executive, supervisor, user
Compatibility bit:PSL<31>	Either PDP-11 or native mode

Figure A-3. ISP Description of VAX cpu

Autoincrement	contents of the register are used as the address of the operand, and then incremented by the size of the operand. If the PC is the specified register, the mode is called the Immediate mode.
Autoincrement deferred mode	contents of the register are used as the address of a location in memory containing the address of the operand, and then are incremented by four (the size of an address). If the PC is the specified register, the mode is called the Absolute mode.
Displacement mode	the value stored in the register is used as a base address. A byte, word, or longword signed constant is added to the base address, and the resulting sum is the effective address of the operand.
Displacement deferred mode	the value stored in the register is used as the base address of a table of addresses A byte, word, or longword signed constant is added to the base address, and the resulting sum is the address of the location that contains the actual address of the operand.

Figure A-4. VAX Addressing Modes

The native and compatibility mode instructions make up an executable instruction set with over 300 different opcodes. Native mode instructions are variable length, use a variety of 5 data types and as stated earlier uses 16 32-bit general purpose registers. Compatibility mode instructions use integer data types and uses 8 16-bit general purpose registers. While native mode is the primary instruction execution state of the machine, and compatibility mode the secondary, the two instruction sets are closely related and programming characteristics are similar. As a result, user processes can execute both native mode and compatibility mode images. In the native mode there are 3 classes of instructions (Ref 21:408):

1) General Data Type Manipulation
   a. Integer and floating-point instructions.
   b. Packed decimal instructions.
   c. Character-string instructions
   d. Bit field instructions

2) Special Data Type/Structure Manipulation
   a. Queue manipulation instructions.
   b. Address manipulation instructions.
   c. User-programmed general register control instructions.

3) Program Flow Control & Procedure Calls

a. Branch, jump, and case instructions

b. Subroutine call instructions

c. Procedure call instructions

The processor recognizes 32 interrupt priority levels, with the highest 16 reserved for hardware generated interrupts and the lowest 16 levels reserved for software requested interrupts (Ref 21:409). When handling interrupts, the processor enters a special system-wide context during which it executes in kernel mode using a special stack called the interrupt stack. The interrupt stack is used only after receipt of an interrupt and thus is not accessible by the user. Interrupt service routines (ISR) are executed at a level corresponding to the associated interrupt. Should a higher priority interrupt occur, the processor will honor that request at its priority level. When the REI (return from exception or interrupt) is issued by the ISR the processor then returns control to the previous level.

Primary Memory

The main memory used in the VAX is connected to the SBI via a memory controller. Physically, it is built using 16K MOS RAM chips, and is organized into 64-bit quadwords plus an 8-bit error correcting code (ECC). Each 64-bit read requires an 800 ns cycle time, while a 64-bit write requires a 1400 ns cycle time. However, the 8K byte

write-through memory cache results in an effective 290 ns memory access time. The minimum memory requirement on the VAX system is 1M-bytes, while maximum memory capacity is 8M-bytes. By adding the MA 780 shared memory option, memory on the VAX 11/780 can be expanded to 12M-bytes (Ref 21:405).

Interleaving is possible with two controllers and equal amounts of memory on each. Interleaving is enabled/disabled under program control. It is performed at the quadword level because of the memory organization. The memory controllers allow the writing of data in full 32-and 64-bit units. Also upon command from an SBI device, individual bytes (or a single byte) may be written. (Ref 21:404).

Each memory controller buffers up to four memory access requests. This request buffering results in an increase in memory throughput, and overall system throughput, while decreasing the need for interleaving for most configurations. Use of the buffer matches memory bandwidth with that of the SBI at 13.3 million bytes/second, primarily because it enables transaction concurrency. For example, the memory controller can except a WRITE command from a MASSBUS adapter while it is reading previously requested data by the processor resulting in increased throughput. If there were no request buffer, there would be about a 50% degradation in memory bandwidth,

making interleaving necessary to approach the bandwidth of the SBI.

The Bus System

As stated earlier, the central connecting component of the VAX system is the Synchronous Backplane Interconnect (SBI). There are 84 signal lines with the data path being 32 bits wide. The remainder of the VAX bus system includes a Physical Address (PA) bus, and a memory data (MD) bus which interface directly with the SBI, and the MASSBUS and UNIBUS subsystems which interface with the SBI via adapters. Finally, there is a Q-bus which serves as the communication path for components of the VAX's Diagnostic Subsystem. The Q-bus is interfaced with the processor through the Writable diagnostic control store as shown in Figure A-1.

Part 2 - The Unix Operating System

The AFIT VAX runs the 'Berkeley UNIX' version 4.1 operating system. This version of UNIX is a highly embellished descendent of the UNIX operating system developed by the Computing Science Research Group at Bell Laboratories in New Jersey during the late 1960's. The version is tagged 'Berkeley' due to the numerous changes made to the standard UNIX system by a programmer group at the University of California at Berkeley (Ref 13:6). Major features of Berkeley UNIX include:

1) 'ex' text editor and 'vi' screen editor
2) C-shell command interpreter/user interface
3) Pascal programming language system
4) Lisp interpreter
5) INGRES data base management system

While these enhanced features serve to set Berkeley UNIX apart from other versions, the basic structure and implementation of the operating system is unchanged. The following discussion will highlight the structural and functional aspects of the UNIX operating system in general.

UNIX is an interpretive, multi-tasking and multi-user operating system. UNIX is composed of three major parts, 1) the kernel, 2) the file system and 3) the shell. The

kernel is the hardware interface portion of UNIX and handles all system resource management, intra-system communication and file system organization. The file system is the organizing structure for data and is probably (from the user's standpoint) the most important part of UNIX. In addition to being a repository for data, the file system provides a means of organizing the layout of data in complex yet efficient ways (Ref 13:3). Finally, the shell is the command interpreter/user interface and is really a system utility that translates user requests into actions required of the kernel or other utility programs.

Kernel Implementation

The UNIX kernel is that portion of the operating system which exercises process control and control of the I/O system. In performing process control, programs and commands submitted by users are executed as user processes until a system function is required. At this point, the system is called as a subroutine and the process execution environment is changed from user to system. The process will remain a system process until the required system function is completed. While the environment changes from the systems viewpoint, the executing process remains the same from the user's viewpoint.

To provide swapping efficiency, processes may execute from a read-only text segment which is shared by all

processes executing the same code. All current read-only segments in the system are maintained from a text table which contains the location of the text segment on secondary memory and a count of the number of processes sharing the segment when it is in primary memory. In the user environment the process has a unique data segment associated with it. This segment is used strictly by the user and has two growing boundaries. One is the process stack and a second is the useable data area which can be increased only by explicit requests from the user. Also associated with the process is a fixed size system data segment which holds information needed by the system when the process is active. This data includes saved cpu registers, open file descriptors, accounting information, scratch data area and a stack for the system phase of the process. Unlike the user data segment, the system data segment can not be addressed by the user. Both user and system data segments are swapped with the process, and with the process, make up a process image. (Ref 4:1971-1980)

In addition to the data maintained for each active process, a process table is maintained which contains information the system needs about inactive processes. This information includes the process name, location of user and system data segments, and scheduling information. The process table entry is allocated when the process is created and freed when the process terminates (Ref

20:1933). Throughout the life of the process, the kernel directly addresses the associated process table entry for entry updates and status checks.

Processes are created by the system primitive "Fork", which creates two nearly identical copies of a process. One is labeled the parent while the second is called the child process. All parts of the parent process image are inherited by the child including open files.

I/O System

The UNIX I/O system consists of two parts--the block I/O system and the character I/O system. Structured I/O system and unstructured I/O system are more appropriate names for the two parts. Each I/O device is characterized by a major and minor device number and a class which indicates whether it is a block or character I/O device. All device drivers are accessed through an array which is indexed by the major device number. The minor device number is then passed to the device driver and allows access to one of the several identical physical devices. The use of this array of entry points to the device drivers, results in only one connection between the system code and the device drivers, and allows easier creation and modification of device drivers. (Ref 20:1937)

The block I/O system device driver is implemented to emulate a model block I/O device on a physical device. The

model block I/O device consists of randomly addressed, secondary memory blocks of 512 bytes (1024 in Berkeley UNIX) each which are uniformly addressed from 0 to N where N is the sized of the device. Block I/O devices are accessed through a layer of buffering software which uses a pool of system buffers, each assigned a device name and address, to perform I/O transactions. All devices that are not in the block I/O system are part of the character I/O system. Whereas, block I/O devices usually include magnetic tape and disks, character I/O devices may include these and all character devices such as communication lines, paper tape and line printers. Unlike block I/O device drivers, character I/O device drivers vary depending on implementation in the techniques used to handle I/O (Ref 20:1939).

File System Implementation

The UNIX system views a file as a one dimensional array of bytes. These files are attached to a hierarchy of directories, which are themselves files that cannot be written by users. Disks used for file storage are viewed as block I/O devices and consequently are manipulated as a randomly addressable array of 512 byte blocks (1024 bytes in Berkeley UNIX). The file system breaks a disk into four self-identifying regions. The first block, located at address 0, is not used by the file system but is instead

used strictly for booting procedures. The second block is known as the super block and contains disk size and boundary information. The third block contains the i-list which is the list of file definitions. Each file definition is a 64-byte structure called an i-node, which contains the device name (major and minor numbers) and an l-number to uniquely identify each file. The remaining blocks make up the free storage blocks which are available for file contents.

The hierarchical structure of the UNIX file system supports three types of files. As stated above, at the top of each level in the hierarchy is a directory file which is a list of other files and directories. The second type of files are ordinary files which are used to store user and system data. Finally, UNIX supports special files which are used primarily in device and system resource management (Ref 20:1941-1943).

When implemented, one file system could use an entire disk, which is usually the case for small disks. However, if the disk is large, it can be split into several logical disks with file systems on each of the logical disks.

The Shell

The third portion of the UNIX system is the shell, which is the command language interpreter. The shell provides the most important communication channel between

the system and its users. Even though the shell allows users to communicate with the system it is not actually part of the operating system. As a result, it enjoys no special privileges and is handled by operating system kernel as a swappable process (Ref 4:1971).

In addition to being the means of interactive user communication, the command language supported by the shell can also be used in user written programs. This enables users to customize their communication interface for the specific type of work being done on the system.

The full language accepted by the shell is complex to the newcomer because it performs a number of functions. However, interactive commands are handled in a straight forward and consistent manner. A command is a sequence of words separated by white space. The first word is always the name of the command, which is an executable file. The remainder of the command line contains one or a more of the following words:

1) simple strings of characters

2) a file name preceded by <, >. >>

3) a string containing a file name expansion character

Simple arguments are passed to a command as an array of strings and thereafter are interpreted by that program. This enables uniformity in the treatment of arguments. The <, >, and >> characters are used to redirect standard input and output. Finally, *, ?, , and [], characters are used to

expand command line file name arguments.

A powerful feature of the shell is its capability to support command line pipes. A pipe is in effect an open file connecting two processes; information written into one end of the pipe may be read from the other end with synchronization, scheduling and buffering handled automatically by the system (Ref 20:1957). A linear array of processes thus becomes a set of coroutines simultaneously processing an I/O stream. The vertical bar is used to separate the various program names used in a pipe.

Conclusion

The VAX/UNIX combination provide a powerful processing environment to the user. The speed and reliability built into the VAX super-mini architecture plus the processing versatility of UNIX allow users working under proper system workload conditions to productively carry out their processing tasks. While users can designate processes to be run in the background, the AFIT SSC is configured exclusively for interactive processing. Thus, should the processing capability of SSC be matched with too large a workload, any degredation in performance will be apparent to system users.

APPENDIX B

Data Collection and Preparation

The data collection and preparation techniques used in this study made maximum use of UNIX performance data sources and data preparation utilities. The plan of attack used for generating data sets for use in the analyses performed was three-fold. First the data available from each of the above sources was reviewed and evaluated for the type of performance information that was available. Secondly, test runs were made to gain familiarization with the formats and sources of the generated data. Finally, monitoring periods and sampling rates were established for the software monitors and file interrogation and data extraction routines were developed for use on system accounting files.

UNIX System Performance Data Sources

The UNIX system provides a number of system accounting and performance monitoring commands to the user. These include:

1) ps (process status)	- process status monitor
2) sa (process accounting)	- accounting data summary
3) ac (login accounting)	- accounting data summary
4) last (process log)	- process exec. monitor
5) vmstat (activity monitor)	- overall system monitor
6) iostat (activity monitor)	- disk activity monitor
7) lastcomm (process log)	- process exec. monitor
8) time (process run timer)	- process exec timer
9) df (disk utilization)	- disk utilization monitor

Of the nine commands listed, three of them ps, df, vmstat, and iostat can be classified as system 'snapshot' commands or software monitors. The output from these commands give status various system components at the time the command was issued. The other commands gather information from system accounting files and provide summary output of accounting data collected for a designated period of time. Each of the commands is explained in fair detail in the on-line documentation, which can be reviewed using 'man' followed by the command of interest.

Data Collection Routines

Based on the data available from each of the above sources, the data collection routine developed incorporated all of the snapshot commands and used only the output from

the system accounting command sa. Two UNIX scripts, datascr.1dayI and datascr.1dayII, were developed to run the software monitors at specified monitoring rates on alternate days. The scripts collected all the output from iostat, vmstat and df for a entire 24 hour period. Thus, at the end of each day three files of software monitor data were created. Because monitoring rates were high for iostat and vmstat (1 sample every 5 seconds) these files grew rapidly and required extensive disk space. As a result, daily dumps were made using the UNIX tape archiver 'tar,' to avoid consuming all alloted disk space.

The system accounting data used was collected using the script 'saptot,' which beginning at midnight, extracted all entries made in the system accounting file \usr\acct during the previous 24 hour period. The extraction was done using a C program 'sap' that interrogated the system accounting data structure used in the accounting file. Initially, the data for the 24 hour period was then broken into files which covered exact thirty minute periods starting at 0000. However, after creation of the first data set it was discovered that the software monitor data which was collected to be summarized in 30-minute intervals was off by as much a eight minutes due monitor lag during times of intense processor activity. Thus, a second accounting data script, 'sapprep' was used for generation of the second data set. With sapprep, accounting data files were

not built until after monitor data files, so that beginning and ending times for intervals of monitored activity could be used in creating accounting files. To do this a C program, 'cdts.c' was developed to convert time-date-month-year inputs into seconds since 0000 1 Jan 1970 (system time epoch used for computing current time). A second C program, 'resap.c,' was used to format the converted times.

Data Preparation Routines

After monitor and accounting data were collected, several routines had to be developed to reduce the extensive amount of monitor data and combine monitor data with accounting data. To perform the necessary data reduction and manipulation, extensive use was made of the 'awk' (report generator) and 'sed' (stream editor) UNIX utilities. The routines developed performed five primary functions:

1. Text Stripping - all non-data text was removed from collected data files.

2. File Splitting - large monitor data files were split into files representing approximate 30 minute intervals of system activity.

3. Data Selection - only specific items of data were used from the system accounting files. Items included averaged cpu, memory and i/o activity for each interval, frequency counts of selected types of processes executed

during each interval, and the time required to execute the 'date' command (used as benchmark to indicate response time).

4. Data Reduction - monitor data files representing 30 minutes of system activity were averaged so that one day of system activity was represented by a file of 48 cases.

5. Data Formatting - reduced monitor data and selected accounting data were combined into a composite data file and formatted.

Along with the data collection routines that follow, are all the routines used for data set preparation. Both C programs and UNIX scripts are documented using a program header. In the case of documented scripts, references to modules called will show any other scripts that are used within that script. In some cases, scripts used within scripts were documented as part the primary script using a mini-header. While this implies that subordinate scripts were contained as part of the primary script during execution, in practice, all scripts used were separate executable files in the same directory.

```
#############################################################################
#   DATE:  18 July 1983
#   VERSION:  1.5
#   NAME:  DATASCR.1DAYI and DATASCR.1DAYII
#   MODULE NUMBER:  1.0
#   FUNCTION:  These are UNIX scripts which were used to collect system
#              performance data from AFIT's VAX 11/780. The script executed
#              system software monitors and redirected output from the
#              monitors into data files.
#   INPUTS:  NONE.
#   OUTPUTS:  Data collection files for four system software monitors.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ:  NONE.
#   FILES WRITTEN: vmdataXXxxx,iodataXXxxx,dfdataXXxxx,psdataXXxxx.
#                  XX = date, xxx=month
#   MODULES CALLED:  nps.
#   CALLING MODULES: NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#############################################################################
#
set maxdays = 1    # set max number of days to run monitors

set days = 1       # initialize day counter

set runs = 288     # set max number of times run monitors per day
         # runs should be computed base on the sampling interval
         # and period set for vmstat and iostat

set x=$days
while ($x <= $maxdays)  # perform data collection loop

set a=$runs
while ($a > 0)

# collect ps data using nps (number of user processes)
cat sep >> psdfI; nps >> psdfI &

# collect vmstat data (memory and overall system activity)
cat sep >> vmstatfI; date >> vmstatfI; vmstat 5 60 >> vmstatfI &

# collect iostat data (disk activity)
cat sep >> iostatfI; date >> iostatfI; iostat 5 60 >> iostatfI &
```

```
# collect df data (disk utilization)
cat sep >> dfstatfI; date >> dfstatfI; df|dfform >> dfstatfI &
wait
@ a-- # decrement collection loop counter
end

# create permanent data files from temp files in while loop}i
echo FILE CREATED: >> vmdatalI
cat sep >>vmdatalI; date >>vmdatalI; cat vmstatfI >>vmdatalI &

echo FILE CREATED: >> iodatalI
cat sep >>iodatalI; date >>iodatalI; cat iostatfI >>iodatalI &

echo FILE CREATED: >> psdatalI
cat sep >>psdatalI; date >>psdatalI; cat psdfI >>psdatalI &

echo FILE CREATED: >> dfdatalI
cat sep >>dfdatalI; date >>dfdatalI; cat dfstatfI >> dfdatalI &
wait

rm vmstatfI iostatfI psdfI dfstatfI &  # remove temp files
wait

mv vmdatalI ~brunsys/Data/daylvmdataI &  # move permanent data files to
mv iodatalI ~brunsys/Data/dayliodataI &  # Data directory of brunsys
mv psdatalI ~brunsys/Data/daylpsdataI &
mv dfdatalI ~brunsys/Data/dayldfdataI &
wait

@ x++  #increment overall loop counter

end
############################################################
#MODULE NAME: NPS
#FUNCTION:  This module counts the number of user
#           processes executing on the system.
#           NOTE: The module used in datascr.ldayI would
#                 be normally located in the directory
#                 from which datascr.ldayI is being
#                 executed. It is shown here only for
#                 documentary purposes.
#date
#echo ____________________
#echo total user procs:
#ps -a | wc -l
#
```

```
#####################################################
#  MODULE NAME:  DATASCR.1DAYII
#  FUNCTION:  This module is identical to script
#             datldayI.scr in function but creates
#             different temporary data files (end in
#             II instead of I).  This script was used
#             on alternating days with datldayI.scr
#             so that during overlap periods of data
#             collection different data files would
#             constructed.
#####################################################
set maxdays = 1
set days = 1
set runs = 288

set x=$days
while ($x <= $maxdays)

set a=$runs
while ($a > 0)

cat sep >> psdfII; nps >> psdfII &
cat sep >> vmstatfII; date >> vmstatfII; vmstat 5 60 >> vmstatfII &
cat sep >> iostatfII; date >> iostatfII; iostat 5 60 >> iostatfII &
cat sep >> dfstatfII; date >> dfstatfII; df|dfform >> dfstatfII &
wait
@ a--
end

echo FILE CREATED: >> vmdatalII
cat sep >>vmdatalII; date >>vmdatalII; cat vmstatfII >>vmdatalII &

echo FILE CREATED: >> iodatalII
cat sep >>iodatalII; date >>iodatalII; cat iostatfII >>iodatalII &

echo FILE CREATED: >> psdatalII
cat sep >>psdatalII; date >>psdatalII; cat psdfII >>psdatalII &

echo FILE CREATED: >> dfdatalII
cat sep >>dfdatalII; date >>dfdatalII; cat dfstatfII >> dfdatalII &
wait

rm vmstatfII iostatfII psdfII dfstatfII &
wait

mv vmdatalII ~brunsys/Data/daylvmdataII &
mv iodatalII ~brunsys/Data/dayliodataII &
mv psdatalII ~brunsys/Data/daylpsdataII &
mv dfdatalII ~brunsys/Data/dayldfdataII &
wait
@ x++
end
```

```
#########################################################################
#   DATE:  20 July 1983
#   VERSION:  1.2
#   NAME:   SAPTOT
#   MODULE NUMBER:  1.0
#   FUNCTION:  This script performs automatic accounting data collec-
#              tion using 'sap.c' and '/etc/sa'.  The system accounting
#              file is scanned for all entries made within the last
#              24 hour period.  These entries are copied to a raw
#              accounting data file which is then broken into 30 minute
#              intervals using sap.  Each of the 48 output files from
#              sap are processed using sa to get summarized accounting
#              data files. Resulting files are based on fixed 30 minute
#              intervals starting at approximately 0000.  They most likey
#              will not directly correspond to 30 minute intervals in
#              software monitor data.
#   INPUTS:  NONE.
#   OUTPUTS:   Summarized accounting data files.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ:  '/usr/acct' system accounting file.
#   FILES WRITTEN: 'saout.*' summarized accounting data files..
#   MODULES CALLED:  NONE.
#   CALLING MODULES:  NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#########################################################################
#
cd /usr/public/brunsys/Dataroutines
set et = `date | systim`
set bt = 0
set period = 86400
set interval = 1800
set fcnt = 0

@ bt = $et - $period
date > DSAP.TOT
sap $bt $et /usr/adm/acct acct.sapred >> DSAP.TOT &

wait
while ($bt < $et)
@ fcnt++
sap $bt `expr $bt + $interval` acct.sapred sapout.$fcnt > DSAP.$fcnt &
@ bt += $interval
end
```

```
wait
while ($fcnt > 0)
date > saout.$fcnt
/etc/sa -aijl sapout.$fcnt >> saout.$fcnt &
@ fcnt--
end

wait
rm sapout.* DSAP.*
mv acct.sapred saout.* /usr/public/brunsys/Data
```

```
/*#############################################################################*
#   DATE: 9 June 1983
#   VERSION: 1.6
#   NAME:  CDTS.C
#   MODULE NUMBER:  1.0
#   FUNCTION:  This program takes a time, date, month and year as argument
#              and computes the number of seconds since the system current
#              time epoch of 0000 1 Jan 1970.  The time can then be used to
#              interrogate system accounting files which makes time entries
#              based on the epoch.
#   INPUTS:  Time Date Month Year (i.e. 183059 24 June 1983)
#   OUTPUTS:  Number of seconds since 0000 1 Jan 1970
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ:  NONE.
#   FILES WRITTEN: NONE.
#   MODULES CALLED: NONE.
#   CALLING MODULES: NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#
*#############################################################################*/
#define     LOCTIME                  500    /* difference from g.m.t */
#define     HRCLOCK                 2400
#define     STARTTIME               0000
#define     STARTDATE                  1
#define     STARTMONTH                 1
#define     STARTYEAR               1970  /* system clock initial year */
#define     LEAPYR1                 1972
#define     MINSECS                   60  /* conversion constants */
#define     HRSECS                  3600
#define     DAYSECS                86400
#define     MONSECS              2592000
#define     YEARSECS            31536000

main (argc,argv)
int argc;
char *argv[];

{
int plusdays;
long time,hours,mins;
long year,years;
long seconds,date,days,m,months;
long secsyear,secsmon,secsday;
long secshr,secsmin,totsecs;
char *month,*tin;
char *marray[12];

plusdays = 0;
```

```
marray[1] = "Jan"; marray[2] = "Feb";   /* Initialize month array */
marray[3] = "Mar"; marray[4] = "Apr";
marray[5] = "May"; marray[6] = "Jun";
marray[7] = "Jul"; marray[8] = "Aug";
marray[9] = "Sep"; marray[10] = "Oct";
marray[11] = "Nov"; marray[12] = "Dec";

tin = argv[1];                              /* store input arguments into  */
                                            /* computational variables and */
time = ((long)atol(tin))/100 + LOCTIME;     /* perform conversions where   */
                                            /* necessary                   */
seconds = ((long)atol(tin))%100;

year = (long)atol(argv[4]);

date = (long)atol(argv[2]);

month = argv[3];

if ((time > HRCLOCK) &&
(date == 1 ) &&
(strcmp(marray[1],month) == 0))
{
time = time - HRCLOCK;
plusdays++;
}

if ((year == LEAPYR1) && (strcmp(marray[1],month)!=0) &&
(strcmp(marray[2],month)!=0))
plusdays++;

else if (((strcmp(marray[1],month)==0) || (strcmp(marray[2],month)==0)) &&
   (year > LEAPYR1))
plusdays = plusdays + 1 + ((year-1) - LEAPYR1)/4;

else if ((strcmp(marray[1],month)!=0) && (strcmp(marray[2],month)!=0) &&
   (year > LEAPYR1))
plusdays = plusdays + 1 + (year - LEAPYR1)/4;

mins = (time - STARTTIME)%100;
hours = ((time - STARTTIME)-mins)/100;
((date==1)&&(strcmp(marray[1],month)==0)) ? (days=0) :(days=date-STARTDATE);

if (strcmp(marray[1],month) == 0)  /* determine month an make necessary */
m = 1;                             /* day count adjustments             */

else if (strcmp(marray[2],month) == 0)
{m = 2; plusdays = plusdays + 1;}

else if (strcmp(marray[3],month) == 0)
{m = 3; plusdays = plusdays - 1;}
```

```
else if (strcmp(marray[4],month) == 0)
{m = 4; (days>23) ? hours-- : (hours=hours);}

else if (strcmp(marray[5],month) == 0)
{m = 5; hours--;}

else if (strcmp(marray[6],month) == 0)
{m = 6; plusdays = plusdays+1; hours--;}

else if (strcmp(marray[7],month) == 0)
{m = 7; plusdays = plusdays+1; hours--;}

else if (strcmp(marray[8],month) == 0)
{m = 8; plusdays = plusdays+2; hours--;}

else if (strcmp(marray[9],month) == 0)
{m = 9; plusdays = plusdays+3; hours--;}

else if (strcmp(marray[10],month) == 0)
{m = 10; plusdays = plusdays+3;
(days<23) ? hours-- : (hours=hours);}

else if (strcmp(marray[11],month) == 0)
{m = 11; plusdays = plusdays+4;}

else if (strcmp(marray[12],month) == 0)
{m = 12; plusdays = plusdays+4;}

else
printf("improper month specified\n");

months = m - STARTMONTH;
years = year - STARTYEAR;

secsmin = mins * MINSECS;
secshr = hours * HRSECS;
secsday = (days+plusdays) * DAYSECS;
secsmon = months * MONSECS;
secsyear = years * YEARSECS;

printf("%d\n",(totsecs=seconds+secsmin+secshr+secsday+secsmon+secsyear));

}
```

```
/*#########################################################################*
#   DATE:  20 July 1983
#   VERSION:  1.5
#   NAME: SAP.C
#   MODULE NUMBER:  1.0
#   FUNCTION:  This program interrogates the system accounting files and
#              extracts accounting data entered between specified begin
#              and end times.
#   INPUTS:  Beginning and ending times to limit data extraction.
#   OUTPUTS:  Accounting file entries which fall within specified time
#             limits provided as inputs.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ:  '/usr/acct' system process accounting file.
#   FILES WRITTEN: Reduced system accounting file..
#   MODULES CALLED:  NONE.
#   CALLING MODULES:  NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
*#########################################################################*/
#include <stdio.h>
#include <ctype.h>
#include <time.h>
#include <utmp.h>
#include <sys/types.h>
#include <sys/timeb.h>

typedef u_short comp_t;

struct  acct
{
char    ac_comm[10];            /* Accounting command name */
comp_t  ac_utime;               /* Accounting user time */
comp_t  ac_stime;               /* Accounting system time */
comp_t  ac_etime;               /* Accounting elapsed time */
time_t  ac_btime;               /* Beginning time */
short   ac_uid;                 /* Accounting user ID */
short   ac_gid;                 /* Accounting group ID */
short   ac_mem;                 /* average memory usage */
comp_t  ac_io;                  /* number of disk IO blocks */
dev_t   ac_tty;                 /* control typewriter */
char    ac_flag;                /* Accounting flag */
};

struct acct selctbuf,*sbfptr;

char  *racctf;
char  *wacctf;

main (argc,argv)
int argc;
```

```
char *argv[];
{
FILE *rf;
FILE *wf;
long begtime;
long endtime;
time_t testtime;
int gtcount,i;

begtime = (long)atol(argv[1]);   /* assign begin and end times provided as */
endtime = (long)atol(argv[2]);   /* arguments                                */
/*  printf("%d %d\n",begtime,endtime); */
testtime = 0;

racctf = argv[3];  /* assign names of accounting files to be read and  */
wacctf = argv[4];  /* written which were provided as arguments         */
if ((rf = fopen(racctf, "r")) == NULL)
{
printf("NO %s\n",racctf);
exit(1);
}
if ((wf = fopen(wacctf, "w")) == NULL)
{
printf("CANNOT CREATE %s\n", wacctf);
exit(1);
}
gtcount = 0;
for (;;)
{
if ((fread((struct acct *)&selctbuf, sizeof(selctbuf), 1, rf) != 1) ||
(gtcount > 250))
break;
testtime = selctbuf.ac_btime;
if (testtime >= begtime && testtime < endtime)
{

/* printf("time=%d ",testtime);
 for (i=0; i<10; i++)
   printf("%c",selctbuf.ac_comm[i]);
 printf("\n"); */

 fwrite((struct acct *)&selctbuf, sizeof(selctbuf), 1, wf);
}
else if (testtime > endtime)
gtcount++;
}
exit(0);
}
```

```
#########################################################################
#   DATE:  17 Aug 1983
#   VERSION:  1.1
#   NAME:  GETTIMS
#   MODULE NUMBER: 1.0
#   FUNCTION:  This script incorporates use of awk and the UNIX stream
#              editor 'sed' to select lines from a file of data col-
#              lection times which correspond to thirty minute inter-
#              vals.  The times which are originally given in the
#              form of the output from the UNIX 'date' command are
#              reformatted by awk for input to 'cdts', a C program
#              which converts a time,date,month and year input into
#              seconds since 0000 1 Jan 1970.  After times are select-
#              ed and written to a file, 'cdts' is written in the file
#              before each time.  The file is then changed to an
#              executable file and executed to produce all times in
#              seconds.
#   INPUTS:  Time files of 'date' output.
#   OUTPUTS:  Files of selected times computed in seconds.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ:  x number of time files.
#   FILES WRITTEN:  'convtimes.x' file of selected times in seconds.
#   MODULES CALLED:  NONE.
#   CALLING MODULES:  NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#########################################################################
#
set fcnt = 1 #initialize file counter
foreach i ($argv)
# reformat date output for cdts
sed -f datsedcoms $i | awk '{print $4,$3,$2,$6}' > ftf

# select times corresponding to 30 minute intervals
sed -e 1d ftf | awk '{if ((NR==1)||((NR-1)%6==0)) print $0 }' > formtimfile

# insert cdts before each selected time
sed -e s/^/cdts' '/ formtimfile > convtimfile

chmod 755 convtimfile  #change mode of file to executable

convtimfile > convtimes.$fcnt  #execute file and redirect output

# rm ftf formtimfile convtimfile #remove temporary files

@ fcnt++
end
```

```
#########################################################################
#   DATE: 15 Aug 1983
#   VERSION:  1.2
#   NAME: RESAP.C
#   MODULE NUMBER:  1.0
#   FUNCTION:  This program constructs a file which will later be
#              changed to an executable file that regenerates ac-
#              counting data files with intervals that better match
#              those in the software monitor data files.
#   INPUTS:  Two inputs must be provided:
#                1.  The name of the raw accounting data file
#                    which will be interrogated by the data
#                    selection program 'sap.c'
#                2.  The name of the file containing adjusted times
#                    (in seconds) of intervals to be used in con-
#                    structing new accounting data files.
#   OUTPUTS:  A file with four entries per line:
#               1. Start time for data selection interval
#               2. End time for data selection interval
#               3. Name of raw accounting file being used
#               4. Number to be used as a temporary file name
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ: File containing adjusted interval times.
#   FILES WRITTEN: Argument portion of the file which will be used
#                  with sap.c to regenerate accounting data files.
#   MODULES CALLED:  NONE.
#   CALLING MODULES:  NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#
#########################################################################
#define    NUMSIZE       10
#define    MAXFILES      48
#define    PERIOD      1800

#include <stdio.h>


main (argc,argv)
int argc;
char *argv[];

{
FILE *rf;
char *rtimef,tbuf[NUMSIZE],tbuflast[NUMSIZE],prevtbuf[NUMSIZE];
char *racctf,*wacctf;
long lasttime,newlasttime;
int  i,n;
```

```
rtimef = argv[1];
racctf = argv[2];
if ((rf = fopen(rtimef, "r")) == NULL)
{
printf("NO %s\n",rtimef);
exit(1);
}

if (fscanf(rf,"%s",tbuf) != EOF)
for(i=0; i<=NUMSIZE; i++)
prevtbuf[i] = tbuf[i];
else
exit(1);

n = 1;
while (fscanf(rf,"%s",tbuf) != EOF)  /* build file of arguments for sap.c */
{
printf("%s %s %s %d\n",prevtbuf,tbuf,racctf,n++);
for(i=0; i<=NUMSIZE; i++)
prevtbuf[i] = tbuf[i];
}
lasttime = atol(tbuf);
newlasttime = lasttime + PERIOD;
printf("%s %d %s %d\n",tbuf,newlasttime,racctf,n++);
exit(0);

}
```

```
/*#########################################################################*
#   DATE:  19 Sep 1983
#   VERSION:  1.0
#   NAME:  SAPPREP
#   MODULE NUMBER:  1.0
#   FUNCTION:  This script combines the function of 'resap.c' with 'awk'
#              and 'sed' to generate accounting data files which coincide
#              with software monitor data files over data collection in-
#              tervals.
#   INPUTS:  Three arguments are required:
#                 argv[1] = Time file for use by resap.
#                 argv[2] = Raw accounting file name.
#                 argv[3] = Directory name for created accounting data files.
#   OUTPUTS:  Directory of accounting data files.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ:   Selected raw accounting files.
#   FILES WRITTEN:  Directory of summarized accounting data files.
#   MODULES CALLED:  resap.c .
#   CALLING MODULES:  NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
*#########################################################################*/
#
# perform resap for given time file and accounting file name
resap $argv[1] $argv[2] > sapprepf1

# add fourth argument required for sap application
cat sapprepf1 | awk '{print $1,$2,$3,"sapout."$4}' > sapprepf2
sed -e s/^/sap' '/ sapprepf2 > sapprocfil

# change file mode to executable and execute it
chmod 755 sapprocfil
sapprocfil

# summarize selected raw accouting files with sa
set fcnt = 1
while ($fcnt < 49)
/etc/sa -aijl sapout.$fcnt > saout.$fcnt
@ fcnt++
end

mkdir $argv[3]
mv saout.* $argv[3]

rm sapprepf? sapprocfil sapout.*
```

```
#########################################################################
#   DATE: 7 Sep 1983
#   VERSION:  1.0
#   NAME:  STRIPTXT
#   MODULE NUMBER: 1.0
#   FUNCTION:  This script uses 'sed' to remove all non-data entries
#              made in collected data files.
#   INPUTS:  Four collected data files - vmdata* psdata* dfdata* saout*.
#            (saout* is a directory of accounting data files).
#   OUTPUTS:  Pure data files with no explanatory text.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ:   Three data files and one directory of files (above).
#   FILES WRITTEN: Three pure data files and a directory of pure data
#                  files 'vmdstrip' 'psdstrip' 'dfdstrip' and
#                  'saoutstrip'.
#   MODULES CALLED:  sedsaout
#   CALLING MODULES:  NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#########################################################################
#
# provide user instruction messages
echo FILE ARGUMENT ORDER MUST =  vmdata*  psdata*  dfdata*  saout*
echo
echo IF ORDER WAS INCORRECT HIT \<BREAK> AND START OVER
sleep 5

echo OUTPUT FILES WILL BE --> vmdstrip psdstrip dfdstrip saoutstrip\(dir\)

# build pure data files

sed -f sedcomms $argv[1] > vmdstrip &
sed -f sedcomms $argv[2] > psdstrip &
sed -f sedcomms $argv[3] > dfdstrip &
sedsaout $argv[4]/[saout.?,saout.??]

cd $argv[4]; rm *; cd ..
rm $argv[1] $argv[2] $argv[3]; rmdir $argv[4]


#####################################################
#  MODULE NAME:  SEDSAOUT
#  FUNCTION:  This script takes as an argument a
#             directory of files and strips non-
#             data entries from each of the files.
#
#
#####################################################
#
mkdir saoutstrip
```

```
set fcnt = 101
foreach i ($argv)
sed -f sedcomms $i > saout.$fcnt
@ fcnt++
end

mv saout.??? saoutstrip

#####################################################
#  MODULE NAME: SEDCOMMS
#  FUNCTION:  These are the 'sed' instructions used
#             to remove text entries made in col-
#             lected data files.
#
#
#####################################################

ls/k//
/F/d
/tin/d
/-/d
/—/d
/__/d
/procs/d
/id/d
/tty/d
/re/s/re   //
/cp/s/cp   //
/avio/s/avio  //
/u/s/u   //
/s/s/s   //
/k/s/k//
/Mon/d
/Tue/d
/Wed/d
/Thu/d
/Fri/d
/Sat/d
/Sun/d
/^$/d
```

```
#############################################################################
#   DATE:  28 July 1983
#   VERSION:  1.0
#   NAME:  RMLN1
#   MODULE NUMBER:  1.0
#   FUNCTION:  This script removes the summary line (line 1) of the
#              vmdata files so that cumulative averages contained in
#              the line will not bias the interval averages that will
#              be computed when the vmdata files are reduced.
#   INPUTS:  Vmdata files.
#   OUTPUTS:  Vmdata files with line 1 removed.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ:  x number of vmdata files.
#   FILES WRITTEN: 'vmd.x' vmdata files with line 1 removed.
#   MODULES CALLED:  NONE.
#   CALLING MODULES:  NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#############################################################################
#
set fcnt = 1
foreach i ($argv)
sed -e 1d $i > vmd.$fcnt
rm $i
@ fcnt++
end
```

```
#########################################################################
#   DATE:  20 Sep 1983
#   VERSION:  1.2
#   NAME:  SPLITDATA
#   MODULE NUMBER:  1.0
#   FUNCTION:   This script uses UNIX utility 'split' to break collected
#               data files from vmstat and df software monitors into
#               files that correspond to 30 minutes of monitored system
#               activity.
#   INPUTS:  Collected vmdata and dfdata files.
#   OUTPUTS:  48 vmdata and 48 dfdata files.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ:  vmdata and dfdata files.
#   FILES WRITTEN: 'vmdxx' 'dfdxx' split files.
#   MODULES CALLED:  NONE.
#   CALLING MODULES:  NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#########################################################################
#
echo FILE ORDER MUST \= dfdstrip vmdstrip
echo IF ORDER INCORRECT\, HIT \<BREAK> AND START OVER
sleep 5

split -6 $argv[1] dfd &
split -360 $argv[2] vmd
rmlnl vmd?? &
# cat dfd?? > dfdcond &
```

```
#########################################################################
#   DATE:  28 July 1983
#   VERSION:  1.0
#   NAME: DFR
#   MODULE NUMBER: 1.0
#   FUNCTION:  This script uses the UNIX utility 'awk' to reduce
#              disk utilization data collected using the 'df' soft-
#              ware monitor.
#   INPUTS:  Files containing df monitor data.
#   OUTPUTS:  Reduced df monitor data.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ: x df data files.
#   FILES WRITTEN: 'dfred' a reduced df data file.
#   MODULES CALLED:  dfcondreduce (awk instruction program)
#   CALLING MODULES:  NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#########################################################################
#
foreach i ($argv)
awk -f dfdcondreduce $i > dfdred
end

#####################################################
#  MODULE NAME:  DFDCONDREDUCE
#  FUNCTION:  This set of awk instructions sums up
#             and averages every six lines of the
#             conditioned df data to give one entry
#             for each 30 minute period of the day
#             monitored.
#
#####################################################

BEGIN{i=1;j=1;}
{
for (j=1; j<=NF; j++)
dftot[j] = dftot[j] + $j;
if ((NR%6)== 0)
{
for (j=1; j<=NF; j++)
{
 dfavg[j] = dftot[j]/6;
 dftot[j] = 0;
}
for (i=1; i<=NF; i++)
printf("%.2f ",dfavg[i]);
printf("\n");
}
}
```

```
#########################################################################
#   DATE:  27 July 1983
#   VERSION:  1.0
#   NAME: PSR
#   MODULE NUMBER:  1.0
#   FUNCTION:  This is an awk script which reduces running process
#              counts in ps data files.  Process counts a reduced
#              from 5 minute to 30 minute interval summaries.
#   INPUTS:  Ps data files containing intervaled counts of running
#            system processes.
#   OUTPUTS:  Reduced ps data file.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ: x number of ps data files.
#   FILES WRITTEN: 'psdred' reduced ps data file(s).
#   MODULES CALLED:  NONE.
#   CALLING MODULES:  NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#########################################################################
#
foreach i ($argv)
cat $i | awk -f psreduce > psdred
end

#####################################################
#  MODULE NAME:  PSREDUCE
#  FUNCTION:  These are the awk instructions which
#             reduce the conditioned ps data file.
#             Five minute interval counts are sum-
#             marized into 30 minute interval counts.
#
#
#####################################################

BEGIN{i=1;}
{
pstot = pstot + $1;
if ((NR%6)== 0)
{
psavg[i] = pstot/6;
pstot = 0;
i++;
}
}END{
for (i=1;i<=48;i++)
printf(" %.2f\n",psavg[i]);
}
```

```
#########################################################################
#   DATE:  27 July 1983
#   VERSION:  1.2
#   NAME:  VMR
#   MODULE NUMBER:  1.0
#   FUNCTION:  This script summarizes collected data from the 'vmstat'
#              software monitor to reflect consecutive 30 minute in-
#              tervals of system activity for a given day.
#   INPUTS:  Pure vmdata files (stripped of text).
#   OUTPUTS:  Summarized vmdata file.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ:  x number of stripped vmdata files.
#   FILES WRITTEN:  'vmdred' reduced vmdata file..
#   MODULES CALLED:  vmreduce
#   CALLING MODULES:  NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#########################################################################
#
foreach i ($argv)
cat $i | awk -f vmreduce >> vmdred
end

#####################################################
#  MODULE NAME:  VMREDUCE
#  FUNCTION:  These are the awk instructions used
#             to summarize a pure vmdata file to
#             reflect 30 minute intervals of sys-
#             tem activity.
#
#
#####################################################

{
if(NF==22)
{
for(n=1;n<=22;n++)
col[n]+=$n
}
else if (length($1)>2)
{
 col[1]+=substr($1,1,2);
 col[2]+=substr($1,3,2);
 for(n=2;n<=21;n++)
   col[n+1]+=$n;
}
else if (length($21)>2)
{
 sz = length($21)-3;
 for(n=1;n<=20;n++)
```

```
    col[n]+=$n;
  col[21]+=substr($21,1,sz);
  st = length(col[21]);
  col[22]+=substr($21,st+1,3);
}

}END{
for(n=1;n<=22;n++)
col[n]/=NR;
for(n=1;n<=22;n++)
printf "%.4f ",col[n] ;
printf "\n"
}
```

```
#########################################################################
#   DATE:  27 July 1983
#   VERSION:  1.0
#   NAME:  IOR
#   MODULE NUMBER:  1.0
#   FUNCTION:  This script is used to reduce collected data from the
#              'iostat' software monitor so that each 30 minutes of
#              activity are summarized.
#   INPUTS:  Conditioned (stripped) iodata file.
#   OUTPUTS:  File of summarized iodata.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ:  x number of collected iodata files.
#   FILES WRITTEN:  'iodred' reduced iodata file.
#   MODULES CALLED:  ioreduce
#   CALLING MODULES:  NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#########################################################################
#
foreach i ($argv)
cat $i | awk -f ioreduce
end

#####################################################
#  MODULE NAME:  IOREDUCE
#  FUNCTION:  These are the awk instructions for
#             summarizing the collected iodata file
#             into 30 minute intervals.
#
#
#####################################################

{
if(NF==12)
{
for(n=1;n<=12;n++)
col[n]+=$n
}
else
{
col[1]+=substr($1,1,2);
col[2]+=substr($1,3,2);
for(n=2;n<=11;n++)
col[n+1]+=$n;
}
}END{
for(n=1;n<=12;n++)
col[n]/=NR;
for(n=1;n<=12;n++)
printf "%.4f ",col[n] ;
```

```
printf "\n"
}
```

```
#########################################################################
#   DATE:  28 July 1983
#   VERSION:  1.0
#   NAME: SCR
#   MODULE NUMBER:  1.0
#   FUNCTION:  This is an UNIX script which incorporates awk to
#              extract the summary line from each accounting data file
#              provided as input.
#   INPUTS:  Accounting data files.
#   OUTPUTS:  File of accounting data file summary lines.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ: x number of accounting data files.
#   FILES WRITTEN:  'sadred' reduced accountin file.
#   MODULES CALLED:  NONE.
#   CALLING MODULES:  NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#########################################################################
#
foreach i ($argv)
cat $i | awk -f sareduce >> sadred
end
```

```
#################################################
#  MODULE NAME:  SAREDUCE
#  FUNCTION:  These are the awk instructions which
#             extract the summary line (line 1) of
#             each entered accounting data file.
#
#
#################################################

{
if (NR==1)
print " ",$1, $2, $3, $4, $5, $6
}
```

```
#########################################################################
#   DATE:  27 July 1983
#   VERSION:  1.0
#   NAME: MCR
#   MODULE NUMBER:  1.0
#   FUNCTION:  This script uses awk to scan the accounting data and
#              compute a count of selected groups of monitored com-
#              mands.
#   INPUTS:  Accounting data files
#   OUTPUTS:  Reduced accounting data consisting of the counts of
#             selected commands.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ: x number of accounting files.
#   FILES WRITTEN: 'mcred' file of monitored command counts.
#   MODULES CALLED:  NONE.
#   CALLING MODULES:  NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#########################################################################
#
foreach i ($argv)
cat $i | awk -f moncomms >> mcred
end

#####################################################
#  MODULE NAME: MONCOMMS
#  FUNCTION: This is the set of awk instructions
#            used to query accounting data files
#            and tally counts of selected commands.
#            The monitored commands are associated
#            with a specific location in array 'mc.'
#            After counting is completed values
#            stored in the array are printed.
#
#
#####################################################

BEGIN{ for (n=1;n<=15;n++) mc[n] = 0;}
{
if (($7=="troff")||($7=="vtroff"))
mc[1]+= $1;

if ($7=="nroff")
mc[2]+= $1;

if ($7=="karel")
mc[3]+= $1;

if (($7=="f77")||($7=="f77pass1"))
mc[4]+= $1;
```

```
if (($7=="vcat.uniq")||($7=="vcat")||($7=="vpr.uniq")||($7=="vpr"))
mc[5]+= $1;

if (($7=="vi")||($7=="ex")||($7=="emacs")||($7=="ed"))
mc[6]+= $1;

if ($7=="cc")
mc[7]+= $1;

if ($7=="slam")
mc[8]+= $1;

if (($7=="pi")||($7=="pix")||($7=="px"))
mc[9]+= $1;

if (($7=="pc")||($7=="pc0")||($7=="pc1")||($7=="pc2"))
mc[10]+= $1;

if (($7=="vmstat")||($7=="iostat")||($7=="ps")||($7=="df"))
mc[11]+= $1;

if (($7=="sa")||($7=="saptot")||($7=="sap"))
mc[12]+= $1;

if (($7=="RUN.S")||($7=="sample")||($7=="replace"))
mc[13]+= $1;

if (($7=="yacc")||($7=="lex"))
mc[14]+= $1;

if (($7=="ls")||($7=="date")||($7=="cat"))
mc[15]+= $1;

}END{
for (n=1;n<=15;n++)
printf "%d ",mc[n];
printf "\n"
}
```

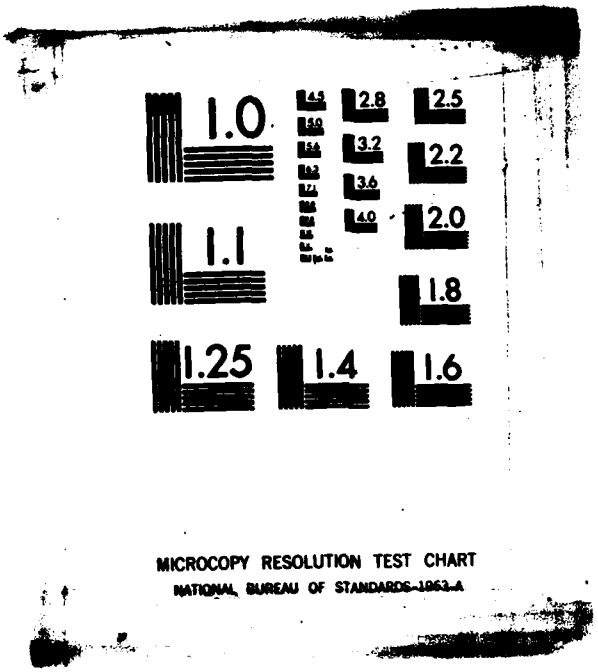
```
#########################################################################
#    DATE:  20 Sep 1983
#    VERSION:  1.0
#    NAME:  REDUCEDATA
#    MODULE NUMBER:  1.0
#    FUNCTION:  This script executes the six data reduction scripts used
#               to reduce pure data files containing software monitor and
#               system accounting data files.
#    INPUTS:  NONE.
#    OUTPUTS:  NONE.
#    GLOBAL VARIABLES USED: NONE.
#    GLOBAL TABLES USED: NONE.
#    LIBRARY ROUTINES: NONE.
#    FILES READ:  All text stripped collected data files.
#    FILES WRITTEN: NONE..
#    MODULES CALLED:  vmr psr mcr dater sar
#    CALLING MODULES:  NONE.
#    AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#########################################################################
#
cat dfd?? >> dfdcond
dfr dfdcond &

vmr vmd.? vmd.?? &

psr psdstrip &

mcr saoutstrip/* &

dater saoutstrip/* &

sar saoutstrip/* &
```

```
/*#####################################################################
#   DATE: 29 July 1983
#   VERSION: 1.4
#   NAME:  CATFLNS.C
#   MODULE NUMBER:  1.0
#   FUNCTION: This program takes as arguments seven files with collected
#             system performance data and concatenates corresponding lines
#             from each file.  The concatenated lines are written to a com-
#             posite data file so that each line of the composite file
#             represents one case from all the collected data.
#   INPUTS: Six files of collected data and the name of the composite
#           data file.
#   OUTPUTS:  A composite data file.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ:  Six files read - argv[1] through argv[6], representing
#                the following collected data files:
#                argv[1] - vmdred (reduced vmstat data)
#                argv[2] - datered (reduced data for date command)
#                argv[3] - psdred  (reduced ps process count data)
#                argv[4] - dfdred  (reduced df disk utilization data)
#                argv[5] - sadred  (reduced accounting file data)
#                argv[6] - mcdred  (reduced monitored processes data)
#   FILES WRITTEN:  One file written - argv[7] which is cmpd, the composite
#                   data file.
#   MODULES CALLED: catstr.
#   CALLING MODULES:  NONE.
#   AUTHOR: LT GREGORY L. BRUNDIDGE
*#####################################################################*/
#define             MAXLN               240  /* max size of input file line */
#define             COMPMAXLN           720  /* max size of combined file line */
#define             NULLSTR            '\0'
#define             NEWLN              '\n'
#define             BLANK               ' '
#define             NUMLINES             48

#include <stdio.h>

main(argc,argv)
int argc;
char *argv[];
{
FILE *fopen(),*cmpfp,*vmfp,*psfp,*bmkfp,*dffp,*safp,*mcfp,*nufp;
register char inln[MAXLN],outln[COMPMAXLN];
int i,l;
```

```
vmfp = fopen(argv[1], "r");   /* assign file descriptors for input files */
bmkfp = fopen(argv[2], "r");
psfp = fopen(argv[3], "r");
dffp = fopen(argv[4], "r");
safp = fopen(argv[5], "r");
mcfp = fopen(argv[6], "r");
cmpfp = fopen(argv[7], "w");
for (l=1; l<=NUMLINES; l++)   /* for each of the input files get a line */
{                             /* and concatenate it with the previous line */
fgets(inln,MAXLN,vmfp);
catstr(outln,inln);

fgets(inln,MAXLN,bmkfp);
catstr(outln,inln);

fgets(inln,MAXLN,psfp);
catstr(outln,inln);

fgets(inln,MAXLN,dffp);
catstr(outln,inln);

fgets(inln,MAXLN,safp);
catstr(outln,inln);

fgets(inln,MAXLN,mcfp);
catstr(outln,inln);

fputs(outln,cmpfp);       /* write concatenated line to composite file*/
for (i=0; i<COMPMAXLN; i++)
outln[i] = NULLSTR;
}
}
/*#####################################################################
# MODULE NAME: CATSTR
# FUNCTION:  This is a modified version of the stringcat function
#            found in C. It takes as arguments two strings and
#            concatenates the first to the last and replaces all
#            newline characters, except the last, with a space.
#####################################################################*/
catstr(s,t)
char s[], t[];
{
int i, j;

i=j=0;
while ((s[i] != NULLSTR) && (s[i] != NEWLN))
i++;
while ((t[j] != NULLSTR) && (t[j] != NEWLN))
s[i++] = t[j++];
s[i++] = BLANK;
s[i++] = NEWLN;
}
```

AD-A138 268 APPLICATIONS OF MULTIVARIATE STATISTICAL TECHNIQUES FOR COMPUTER PERFORMANCE EVALUATION(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI... 3/3

UNCLASSIFIED G L BRUNDIDGE DEC 83 AFIT/GCS/EE/83D-4 F/G 12/1 NL



```
#########################################################################
#   DATE:  3 Aug 1983
#   VERSION:  1.3
#   NAME:  FIXSCRPT
#   MODULE NUMBER:  1.0
#   FUNCTION:  This script formats the composite data file created from
#              the 6 collected data files using 'catflns7.'  Columns
#              are separated by a single space and the 51 entries per
#              line are broken into x lines for use by data analysis
#              programs.
#   INPUTS:  A composite data file.
#   OUTPUTS:  Formatted composite data file.
#   GLOBAL VARIABLES USED: NONE.
#   GLOBAL TABLES USED: NONE.
#   LIBRARY ROUTINES: NONE.
#   FILES READ:  'compdset*' composite data files.
#   FILES WRITTEN: 'compdset*fix' formatted data file.
#   MODULES CALLED:  fixcompdset
#   CALLING MODULES:  NONE.
#   AUTHOR: CAPT GREGORY L. BRUNDIDGE
#
#########################################################################
#
set fcnt = 1
foreach i ($argv)
cat $i | awk -f fixcompdset > compdset.fcnt
@ fcnt++
end
@ fcnt--
cat compdset.* >> DAYS.fcnt &


#####################################################
#  MODULE NAME:  FIXCOMPDSET
#  FUNCTION:  These are the awk instructions used
#             to format the composite data file.
#
#
#####################################################

{
printf "    "NR" "
for (i=1;i<=7;i++)
printf $i" "
printf "\n"

for (i=8;i<=15;i++)
printf $i" "
printf "\n"
```

```
for (i=16;i<=22;i++)
printf $i" "
printf "\n"

for (i=23;i<=30;i++)
printf $i" "
printf "\n"

for (i=31;i<=36;i++)
printf $i" "
printf "\n"

for (i=37;i<=51;i++)
printf $i" "
printf "\n"

}
```

Bibliography

1. Agrawala, A. K. and J. M. Mohr. "Some Results of the Clustering Approach to Workload Modelling," Proceedings of the Thirteenth Meeting of the Computer Performance Evaluation Users Group, 23 - 28, 1977.

2. Anderberg, Michael R. Cluster Analysis for Applications. New York: Academic Press, 1973.

3. Baer, Jean-Loup. Computer Systems Architecture. Rockville, Maryland: Computer Science Press Inc., 1980, pp 429-487.

4. Bourne, S. R., "The UNIX Time-Sharing System: The UNIX Shell," The Bell System Technical Journal, July-August 1978, pp 1971-1989.

5. Bourne, S. R. The UNIX System. Reading, Massachusetts: Addison-Wesley Publishing Company, 1982.

6. Eckhouse, R. H., and Levy, Henry M. Computer Programming and Architecture, The VAX-11. Bedford, Massachusetts: Digital Press, 1980, pp 13-54.

7. Ferrari, Domenico. Computer Systems Performance Evaluation. Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1978.

8. Hartrum, Thomas C. and Jimmy W. Thompson. "The Application of Clustering Techniques to Computer Performance Modelling," Proceedings of the Fifteenth Meeting of the Computer Performance User's Group, October 1979.

9. Hrishikesh, D. Vinod and Aman, Ullah. Recent Advances in Regression Methods. New York: Marcel Dekker, Inc., 1981.

10. Gomaa, H. "A Modelling Approach to the Evaluation of Computer System Performance," 171-179. Modelling and Performance Evaluation of Computer Systems. North Holland Publishing Company, 1976.

11. Magavero, Gregory. A Study of Multivariate Statistical Analysis Techniques for Computer Performance Evaluation, Masters thesis. Air Force Institute of Technology, Wright-Patterson A.F.B., Ohio, 1982.

12. McGilton, Henry and Morgan, Rachel. Introducing the UNIX System, McGraw-Hill Book Company, New York, NY., 1983, pp 1-129,415-540.

13. McIlroy, M. D., Pinson, E. N., and Tague, B. A. "Unix Time-Sharing System: Foreword," The Bell System Technical Journal, July-August 1978, pp 1899-1904.

14. McNichols, Charles W. An Introduction to: Applied Multivariate Data Analysis, unpublished text. School of Engineering, Air Force Institute of Technology, Wright Patterson A.F.B., Ohio, 1980.

15. Neter, John and Wasserman, William. Applied Linear Statistical Models. Homewood, Illinois: Richard D. Irwin, Inc., 1974.

16. Ritchie, D. M. and Thompson, K., "The UNIX Time-Sharing System," The Bell System Technical Journal, July-August 1978, pp 1905-1928.

17. Stopher, Peter R. and Meyburg, Arnim H. Survey Sampling and Multivariate Analysis for Social Scientist and Engineers. Lexington, Massachusetts: Lexington Books, 1979.

18. Stover, Margaret A. Application of Statistical Analysis Techniques to Computer Performance Evaluation, Masters thesis. Air Force Institute of Technology, Wright-Patterson A.F.B., Ohio, 1981.

19. Swarz, Robert S. "Reliability and Maintainability Enhancements for VAX 11/780," The Eighth Annual International Conference on Fault-Tolerant Computing, June 21-23,1978, pp 24-28.

20. Thompson, K., "The UNIX Time-Sharing System: UNIX Implementation," The Bell System Technical Journal, July-August 1978, pp 1931-1945.

21. DEC VAX-11 Systems, DATAPRO RESEARCH CORPORATION (DRC), Delran,NJ.,1982, pp401-412.

22. 'S' A Language and System for Data Analysis, Bell Laboratories, January, 1981.

23. BMDP Statistical Software 1981, Berkeley, California: University of Berkeley Press, 1981.

24. SPSS Statistical Package for the Social Sciences, Second Edition, New York: McGraw-Hill Book Company, 1975.

VITA

Gregory L. Brundidge was born on 23 September, 1957 in Panama City, Florida to Mr and Mrs Joe T. Brundidge. He graduated from Bay High School, Panama City, Florida, in 1975. In 1979 he graduated from the United States Air Force Academy with a Bachelor of Science in Biological Sciences. After graduation he served as an Admissions Advisor for the Academy's Minority Affairs Division. He then attended the Communications Officer Course at Keesler Air Force Base, Mississippi. Upon completion of the course, he was assigned to Tactical Communications Division (TCD), at Langley Air Force Base, Virginia as a Communication Systems Maintenance Officer. While stationed at Langley, he married the former Diane Clayton of Panama City, Florida. He served as a fixed communications maintenance manager for TCD until entering the School of Engineering, Air Force Institute of Technology, in June 1982.

Permanent address: 1121 Wilson Avenue
Panama City, Florida 32401

***

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A138268

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GCS/EE/83D-4	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): See Box 19	
12. PERSONAL AUTHOR(S): Gregory L. Prundidge, B.S., Capt, USAF	
13a. TYPE OF REPORT: MS Thesis; 13b. TIME COVERED: FROM _____ TO _____; 14. DATE OF REPORT (Yr., Mo., Day): 1983 December; 15. PAGE COUNT: 195	
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB. GR.
09	02	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Computer Performance Evaluation, Multivariate Analysis, Statistical Analysis, Statistical Modeling, Performance Modeling, Workload Modeling

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: Applications of Multivariate Statistical Techniques For Computer Performance Evaluation

Thesis Chairman: Thomas C. Hartrum, Ph.D.

Approved for public release: IAW AFR 190-17.
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (AIC)
Wright-Patterson AFB OH 45433
7 Feb 84

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Thomas C. Hartrum, Ph.D.	22b. TELEPHONE NUMBER (Include Area Code): 513-255-2024; 22c. OFFICE SYMBOL: AFIT/ENG

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

In many situations the computer performance evaluation (CPE) analyst has collected an abundance of computer system performance data from the target system's accounting files and software monitors. Traditionally, regression analysis provided the primary means of examing CPE data sets, with the emphasis being on modeling specific workload and performance parameters. Multivariate analysis techniques provide the analyst with additional analysis tools for the examination of relationships, dimension, and structure of large amounts of data. This study examines possible CPE applications for four multivariate analysis techniques. The techniques studied include: Canonical Correlation, Factor Analysis, Discriminant Analysis, and Cluster Analysis. Also included in the study was the use of ordinary least squares regression modeling and ridge regression modeling, to exemplify the traditional problems encountered with use of regression analysis. Depending on the performance evaluation requirements, one or more of the multivariate techniques or ridge regression could be used to perform a preliminary or supplementary CPE data analysis.

END

FILMED

3-84

DTIC